Graduate School ETD Form 9
(Revised 12/07)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Ian Hadley Bell

Entitled
Theoretical and Experimental Analysis of Liquid Flooded Compression in Scroll Compressors

For the degree of Doctor of Philosophy

Is approved by the final examining committee:

Eckhard A. Groll
Co-Chair

James E. Braun
Co-Chair

Galen B. King

W. Travis Horton

To the best of my knowledge and as understood by the student in the *Research Integrity and Copyright Disclaimer (Graduate School Form 20)*, this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Approved by Major Professor(s): Eckhard A. Groll

James E. Braun

Approved by: David C. Anderson 04/04/2011
Head of the Graduate Program Date

School of Mechanical Engineering

Graduate School Form 20
(Revised 9/10)

PURDUE UNIVERSITY
GRADUATE SCHOOL

Research Integrity and Copyright Disclaimer

Title of Thesis/Dissertation:

Theoretical and Experimental Analysis of Liquid Flooded Compression in Scroll Compressors

For the degree of Doctor of Philosophy

I certify that in the preparation of this thesis, I have observed the provisions of *Purdue University Executive Memorandum No. C-22,* September 6, 1991, *Policy on Integrity in Research.**

Further, I certify that this work is free of plagiarism and all materials appearing in this thesis/dissertation have been properly quoted and attributed.

I certify that all copyrighted material incorporated into this thesis/dissertation is in compliance with the United States' copyright law and that I have received written permission from the copyright owners for my use of their work, which is beyond the scope of the law. I agree to indemnify and save harmless Purdue University from any and all claims that may be asserted or that may arise from any copyright violation.

Ian Hadley Bell

Printed Name and Signature of Candidate

04/04/2011

Date (month/day/year)

*Located at http://www.purdue.edu/policies/pages/teach_res_outreach/c_22.html

THEORETICAL AND EXPERIMENTAL ANALYSIS OF LIQUID FLOODED COMPRESSION IN SCROLL COMPRESSORS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Ian Hadley Bell

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

May 2011

Purdue University

West Lafayette, Indiana

To my family

ACKNOWLEDGMENTS

I couldn't have made it to this point without the help of many people.

To begin with, I'd like to thank Andy and Forbes Pearson who took a chance on a college freshman and sparked the fire in me to study refrigeration. Your generosity is in no small part the reason for this dissertation.

I have had the privilege of working with two of the best advisors that I have ever met. Eckhard and Jim, thank you. The caring that you demonstrate for your students is a model that I will take with me wherever I go. Thank you also for all the extra-curricular opportunities that you offered me during my time at Purdue. Galen and Travis, thank you as well for all your help and insightful comments.

I would also like to thank Fritz, Bob, Gilbert and Frank in the shop at Herrick Labs for their support. Special thanks to Frank Lee who was always good for a smile, a laugh, and a helping hand. My experimental work was a success in no small part due to your help.

And of course to my friends and labmates who made living in West Lafayette possible: Ananya, Abhinav, Bryce, Carolin, Christian, Cord, Craig, David, Gerhard, Jitendra, Joe, Jonathan, Margaret, Mallory, Matt, Mayen, Ping, Sebastian, Shaunak, Shiva, Stefan, and a host of others.

Last but not least, my family - Mom, Dad, Nate, Tad, and Abby. You guys have been there through good times and bad, and I thank you for it.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

SYMBOL	UNITS	DESCRIPTION
a	-	Slope at inner starting angle
A	m^2	Area
A_{d-dd}	m^2	Flow area between d_1 and dd chambers
A^*_{flank}	m^2	Effective flank leakage area
A_{free}	m^2	Free area of discharge port
A_g	m^2	Cross-sectional area of gas
A_{inlet}	m^2	Inlet area
$A_{intersection}$	m^2	Port blockage area
A_{max}	m	Maximum oscillation amplitude
A^*_{radial}	m^2	Effective radial leakage area
A_{s-sa}	m^2	Flow area between s_1 and sa chambers
A_{th}	m^2	Throat area
A^*_{total}	m^2	Effective total leakage area
B	rad	S-SA correction term
B'	rad	Derivative of S-SA correction term w.r.t. θ
C_d	-	Discharge coefficient
c_l	kJ/kg-K	Liquid specific heat
COP	-	Coefficient of Performance
COP_{AC}	-	Coefficient of Performance in cooling mode
COP_{base}	-	Coefficient of Performance of baseline system
$COP_{flooded}$	-	Coefficient of Performance of flooded system
COP_{HP}	-	Coefficient of Performance in heat pump mode
c_p	kJ/kg-K	Specific heat
C_{ratio}	-	Ratio of oil/gas capacitance rates

$c_{p,m}$	kJ/kg-K	Constant-pressure mixture specific heat
$c_{v,m}$	kJ/kg-K	Constant-volume mixture specific heat
c_x	m	x-coordinate of the centroid
$c_{x,c1,\alpha}$	m	x-coordinate of the centroid α-th c_1 chamber
$c_{x,c2,\alpha}$	m	x-coordinate of the centroid α-th c_2 chamber
$c_{x,d1}$	m	x-coordinate of the centroid of d_1
$c_{x,d2}$	m	x-coordinate of the centroid of d_2
$c_{x,dd}$	m	x-coordinate of the centroid of dd
$c_{x,I,d1}$	m	x-coordinate of the centroid of "I" part of d_1
$c_{x,I,s1}$	m	x-coordinate of the centroid of "I" part of s_1
$c_{x,Ia,d1}$	m	x-coordinate of the centroid of "Ia" part of d_1
$c_{x,Ia,s1}$	m	x-coordinate of the centroid of "Ia" part of s_1
$c_{x,Ib,d1}$	m	x-coordinate of the centroid of "Ib" part of d_1
$c_{x,Ib,s1}$	m	x-coordinate of the centroid of "Ib" part of s_1
$c_{x,Ic,d1}$	m	x-coordinate of the centroid of "Ic" part of d_1
$c_{x,Ic,s1}$	m	x-coordinate of the centroid of "Ic" part of s_1
$c_{x,Id,d1}$	m	x-coordinate of the centroid of "Id" part of d_1
$c_{x,O,d1}$	m	x-coordinate of the centroid of "O" part of d_1
$c_{x,O,s1}$	m	x-coordinate of the centroid of "O" part of s_1
$c_{x,s1}$	m	x-coordinate of the centroid of s_1
$c_{x,s2}$	m	x-coordinate of the centroid of s_2
c_y	m	y-coordinate of the centroid
$c_{y,c1,\alpha}$	m	y-coordinate of the centroid α-th c_1 chamber
$c_{y,c2,\alpha}$	m	y-coordinate of the centroid α-th c_2 chamber
$c_{y,d1}$	m	y-coordinate of the centroid of d_1
$c_{y,d2}$	m	y-coordinate of the centroid of d_2
$c_{y,dd}$	m	y-coordinate of the centroid of dd
$c_{y,I,d1}$	m	y-coordinate of the centroid of "I" part of d_1
$c_{y,I,s1}$	m	y-coordinate of the centroid of "I" part of s_1

$c_{y,Ia,s1}$	m	y-coordinate of the centroid of "Ia" part of s_1
$c_{y,Ia,d1}$	m	y-coordinate of the centroid of "Ia" part of d_1
$c_{y,Ib,d1}$	m	y-coordinate of the centroid of "Ib" part of d_1
$c_{y,Ib,s1}$	m	y-coordinate of the centroid of "Ib" part of s_1
$c_{y,Ic,d1}$	m	y-coordinate of the centroid of "Ic" part of d_1
$c_{y,Ic,s1}$	m	y-coordinate of the centroid of "Ic" part of s_1
$c_{y,Id,d1}$	m	y-coordinate of the centroid of "Id" part of d_1
$c_{y,O,d1}$	m	y-coordinate of the centroid of "O" part of d_1
$c_{y,O,s1}$	m	y-coordinate of the centroid of "O" part of s_1
$c_{y,s1}$	m	y-coordinate of the centroid of s_1
$c_{y,s2}$	m	y-coordinate of the centroid of s_2
C_h	kW/K	Hot stream capacitance rate
C_{min}	kW/K	Minimum capacitance rate
d	m	Distance between centers of arcs in discharge region
D	m	Diameter
D_h	m	Hydraulic diameter
D_{port}	m	Diameter of discharge port
D_{wall}	m	Diameter of outer wall for compressor
E	kJ/kg	Energy
$\dot{E}_{comp}$	kW	Exergetic destruction from compressor
$\dot{E}_{evap}$	kW	Exergetic destruction from evaporator
$\dot{E}_{hyd-exp,g}$	kW	Exergetic destruction from gas in hydraulic expander
$\dot{E}_{hyd-exp,l}$	kW	Exergetic destruction from liquid in hydraulic expander
$\dot{E}_{leakage}$	kW	Exergetic destruction from leakage
$\dot{E}_{mechanical}$	kW	Exergetic destruction from mechanical losses
$\dot{E}_{mixer}$	kW	Exergetic destruction from mixer
$\dot{E}_{oil-cooler,l}$	kW	Exergetic destruction from liquid in oil cooler
$\dot{E}_{oil-cooler,g}$	kW	Exergetic destruction from gas in oil cooler
$\dot{E}_{reject}$	kW	Exergetic destruction from heat rejection

$\dot{E}_{Reg}$	kW	Exergetic destruction from regenerator
$\dot{E}_{suction}$	kW	Exergetic destruction from suction pressure drop
$\mathbf{f}$	-	System of ODE
f	-	Friction factor
f	Hz	Frequency
f_{xA}	-	Function for calculation of x-coordinate of centroid
f_{yA}	-	Function for calculation of y-coordinate of centroid
$\mathbf{F}_{axial}$	kN	Axial force
$\mathbf{F}_{c1,\alpha}$	kN	Force from α-th c_1 chamber
$\mathbf{F}_{c2,\alpha}$	kN	Force from α-th c_2 chamber
$\mathbf{F}_{CV}$	kN	Force on orbiting scroll from given control volume
$\mathbf{F}_{dd}$	kN	Force from dd chamber
$\mathbf{F}_{dd,a1}$	kN	Force from arc 1 of dd chamber
$\mathbf{F}_{dd,a2}$	kN	Force from arc 2 of dd chamber
$\mathbf{F}_{dd,involute}$	kN	Force from involute part of dd chamber
$\mathbf{F}_{dd,line}$	kN	Force from line of dd chamber
$\mathbf{F}_{oi}$	kN	Force from orbiting inner involute
$\mathbf{F}_{oo}$	kN	Force from orbiting outer involute
$\mathbf{F}_{rad}$	kN	Radial force
$\mathbf{F}_{s1}$	kN	Force from s_1 chamber
$\mathbf{F}_{s2}$	kN	Force from s_2 chamber
$\mathbf{F}_{sa}$	kN	Force from sa chamber
F	-	Flank flow enhancement factor
G	kg/s-m^2	Mass flux
h	kJ/kg	Specific enthalpy
$h_1..h_{11}$	kJ/kg	Specific enthalpy of mixture at cycle state point
$h_{1,g}..h_{11,g}$	kJ/kg	Specific enthalpy of gas at cycle state point
$h_{1,l}..h_{11,l}$	kJ/kg	Specific enthalpy at liquid cycle state point
$h_{2s,inj}$	kJ/kg	Isentropic enthalpy of injected mixture

h_c	kW/m^2-K	Convective heat transfer coefficient
h_{disc}	kJ/kg	Discharge specific enthalpy
h_f	kJ/kg	Upstream enthalpy for flow path
h_g	kJ/kg	Gas specific enthalpy
$h_{g,in}$	kJ/kg	Inlet gas specific enthalpy
$h_{g,out}$	kJ/kg	Outlet gas specific enthalpy
h_{in}	kJ/kg	Inlet specific enthalpy
h_l	kJ/kg	Liquid specific enthalpy
$h_{l,in}$	kJ/kg	Inlet liquid specific enthalpy
$h_{l,out}$	kJ/kg	Outlet liquid specific enthalpy
h_m	kJ/kg	Mixture specific enthalpy
h_{out}	kJ/kg	Outlet specific enthalpy
h_s	m	Height of scroll
h_{sat}	kJ/kg	Specific enthalpy at saturation
$\hat{i}$	-	Unit normal vector in x direction
i	-	Index of step
j_g	m/s	Superficial gas velocity
$\mathbf{J}$	-	Jacobian matrix
$\hat{j}$	-	Unit normal vector in y direction
$\hat{k}$	-	Unit normal vector in z direction
$\mathbf{k}_1$, $\mathbf{k}_2$,...	-	Vector for step
k	-	Index of conjugate angle
k^*	-	Effective ratio of specific heats
k_g	kW/m-K	Gas thermal conductivity
k_l	kW/m-K	Liquid thermal conductivity
k_m	kW/m-K	Mixture thermal conductivity
K	-	Slip ratio
K_e	-	Effective slip ratio
L	m	Length of flow path

L	m	Arclength of frontal area of radial leakage
L	m	Length of line segment forming dd chamber
L_ϕ	m	Distance from base circle to involute
L_i	m	Distance from base circle to inner involute
L_o	m	Distance from base circle to outer involute
$\dot{m}$	kg/s	Mass flow rate
$\dot{m}_{flank}$	kg/s	Flank mass flow rate
$\dot{m}_{gas}$	kg/s	Mass flow rate of gas
$\dot{m}_{g,s}$	kg/s	Mass flow rate of gas solved in oil
$\dot{m}_{g,v}$	kg/s	Mass flow rate of gas in vapor phase
$\dot{m}_{l,hot}$	kg/s	Mass flow rate of oil in hot loop
$\dot{m}_{l,cold}$	kg/s	Mass flow rate of oil in cold loop
$\dot{m}_{m,meas}$	kg/s	Measured mixture mass flow rate
$\dot{m}_{nozzle}$	kg/s	Isentropic nozzle mass flow rate of gas
$\dot{m}_{oil}$	kg/s	Mass flow rate of oil
$\dot{m}_{\text{ref}}$	kg/s	Mass flow rate of refrigerant
$\dot{m}_{radial}$	kg/s	Radial mass flow rate
$\dot{m}_{total}$	kg/s	Total mass flow rate through compressor
m_{CV}	kg	Mass contained in a control volume
m_{d1}	kg	Mass in d_1 chamber
m_{d2}	kg	Mass in d_2 chamber
m_{dd}	kg	Mass in dd chamber
m_{ddd}	kg	Mass in ddd chamber
m_l	kg	Mass of liquid
M	-	Ratio of nozzle to detailed mass flow rates
$\mathbf{M}_{ot}$	kN-m	Overturning moment
$\mathbf{M}_{\bigcirc,c1,\alpha}$	kN-m	Crank pin moment from $c_{1,\alpha}$ chamber
$\mathbf{M}_{\bigcirc,c2,\alpha}$	kN-m	Crank pin moment from $c_{2,\alpha}$ chamber
$\mathbf{M}_{\bigcirc,d1}$	kN-m	Crank pin moment from d_1 chamber

$\mathbf{M}_{\bigcirc,d2}$	kN-m	Crank pin moment from d_2 chamber
$\mathbf{M}_{\bigcirc,dd}$	kN-m	Crank pin moment from dd chamber
$\mathbf{M}_{\bigcirc,dd,a1}$	kN-m	Crank pin moment from arc 1 of dd chamber
$\mathbf{M}_{\bigcirc,dd,a2}$	kN-m	Crank pin moment from arc 2 of dd chamber
$\mathbf{M}_{\bigcirc,dd,involute}$	kN-m	Crank pin moment from involute of dd chamber
$\mathbf{M}_{\bigcirc,dd,line}$	kN-m	Crank pin moment from line of dd chamber
$\mathbf{M}_{\bigcirc,s1}$	kN-m	Crank pin moment from s_1 chamber
$\mathbf{M}_{\bigcirc,s2}$	kN-m	Crank pin moment from s_2 chamber
$\mathbf{M}_{\bigcirc,sa}$	kN-m	Crank pin moment from sa chamber
M_g	g/mol	Gas mole mass
M_l	g/mol	Liquid mole mass
n	-	Number of points in polygon
$\mathbf{n}$	-	Normal vector
$\mathbf{n}_{fi}$	-	Normal vector towards fixed inner involute
$\mathbf{n}_{fo}$	-	Normal vector towards fixed outer involute
$\mathbf{n}_{oi}$	-	Normal vector towards orbiting inner involute
$\mathbf{n}_{oo}$	-	Normal vector towards orbiting outer involute
$\mathbf{n}_x$	-	x-component of normal vector
$\mathbf{n}_y$	-	y-component of normal vector
N	-	Number of points per rotation
N	-	Number of points per temperature bin
N_c	-	Number of pairs of compression chambers
$N_{c,max}$	-	Max number of pairs of compression chambers
N_{comp}	rev/min	Compressor rotational speed
N_{CV}	-	Number of control volumes
N_{exp}	rev/min	Expander rotational speed
N_{flank}	-	Number of flank contact points
$N_{hyd.exp.}$	rev/min	Hydraulic expander rotational speed
N_{pump}	rev/min	Pump rotational speed

p	kPa	Pressure
$p_1 ... p_{11}$	kPa	Pressure at cycle state point
$p_{c1,\alpha}$	kPa	Pressure of α-th c_1 chamber
$p_{c2,\alpha}$	kPa	Pressure of α-th c_2 chamber
p_{crit}	kPa	Critical pressure of refrigerant
p_{CV}	kPa	Pressure for given control volume
p_d	kPa	Discharge pressure
p_{d1}	kPa	Pressure of d_1 chamber
p_{d2}	kPa	Pressure of d_2 chamber
p_{dd}	kPa	Pressure of dd chamber
p_{ddd}	kPa	Pressure of ddd chamber
p_{down}	kPa	Downstream pressure of flow path
p_{evap}	kPa	Evaporation pressure
p_{gc}	kPa	Gas cooler pressure
p_{ratio}	-	Pressure ratio
p_s	kPa	Suction pressure
p_{shell}	kPa	Shell pressure
p_{up}	kPa	Upstream pressure of flow path
P	m	Perimeter
Pr_m	-	Mixture Prandtl number
$\dot{Q}$	kW	Heat transfer rate
$\dot{Q}_{amb}$	kW	Heat transfer with ambient
$\dot{Q}_{cold}$	kW	Heat input in cold loop
$\dot{Q}_{evap}$	kW	Heat transfer in evaporator
$\dot{Q}_{exhaust}$	kW	Heat transfer in exhaust
$\dot{Q}_{inlet}$	kW	Heat transfer in inlet
$\dot{Q}_{oil-cooler,l}$	kW	Liquid heat transfer in oil cooler
$\dot{Q}_{oil-cooler,g}$	kW	Gas heat transfer in oil cooler
$\dot{Q}_{out}$	kW	Heat transfer in condenser

$\dot{Q}_{plates}$	kW	Heat transfer with plates
$\dot{Q}_{reject}$	kW	Heat rejection
$\dot{Q}_{scrolls}$	kW	Heat transfer with scrolls
$\overline{\dot{Q}_{scrolls}}$	kW	Mean heat transfer with scrolls
$\dot{Q}_{wall}$	kW	Heat transfer with wall
$\mathbf{r}$	m	Direction vector
$\mathbf{r}_1$	m	Vector from origin to base circle
$\mathbf{r}_2$	m	Vector from base circle to involute
$\mathbf{r}_{\bigcirc,c1,\alpha}$	m	Vector from crank pin to involute for $c_{1,\alpha}$ chamber
$\mathbf{r}_{\bigcirc,c2,\alpha}$	m	Vector from crank pin to involute for $c_{2,\alpha}$ chamber
$\mathbf{r}_{cent}$	m	Vector from orbiting origin to centroid
$\mathbf{r}_{\bigcirc,s1}$	m	Vector from crank pin to involute for s_1 chamber
$\mathbf{r}_{\bigcirc,s2}$	m	Vector from crank pin to involute for s_2 chamber
$\mathbf{r}_{\bigcirc,d1}$	m	Vector from crank pin to involute for d_1 chamber
$\mathbf{r}_{\bigcirc,d2}$	m	Vector from crank pin to involute for d_2 chamber
r_{a1}	m	Radius of arc 1 in discharge region
r_{a2}	m	Radius of arc 2 in discharge region
$r_{a2,max}$	m	Maximum radius of arc 2 in discharge region
r_b	m	Radius of base circle
r_c	m	Effective radius of chamber
r_{HT}	kW	Lump energy balance residual
r_o	m	Orbiting radius
R	kJ/kg-K	Specific ideal gas constant
Re	-	Reynolds number
Re_{D_h}	-	Reynolds number based on hydraulic diameter
$s_1..s_{11}$	kJ/kg-K	Specific entropy of mixture at cycle state point
$s_{1,g}..s_{11,g}$	kJ/kg-K	Specific entropy of gas at cycle state point
$s_{1,l}..s_{11,l}$	kJ/kg-K	Specific entropy at liquid cycle state point
s_g	kJ/kg-K	Gas specific entropy

s_l	kJ/kg-K	Liquid specific entropy
s_m	kJ/kg-K	Mixture specific entropy
s_{radial}	m	Radial gap flow arclength
SG	-	Specific Gravity
St	-	Strouhal number
$t_{a1,1}$	rad	Angle 1 for arc 1 in discharge region
$t_{a1,2}$	rad	Angle 2 for arc 1 in discharge region
$t_{a2,1}$	rad	Angle 1 for arc 2 in discharge region
$t_{a2,2}$	rad	Angle 2 for arc 2 in discharge region
t_s	m	Thickness of scroll wrap
T	K, °C	Temperature
T_0	K	Ambient temperature
$T_1 ... T_{11}$	K	Temperature at cycle state point
T_{amb}	°C	Ambient temperature
$T_{c,i}$	C	Cold stream inlet temperature
T_{cond}	°C	Condensing temperature
T_{CV}	K	Temperature of a given control volume
T_d	K	Discharge temperature
$T_{dew,s}$	°C	Dew temperature at suction pressure
$T_{dew,d}$	°C	Dew temperature at discharge pressure
$T_{d,isen}$	K	Isentropic discharge temperature
T_{d1}	K	Temperature of d_1 chamber
T_{d2}	K	Temperature of d_2 chamber
T_{dd}	K	Temperature of dd chamber
T_{ddd}	K	Temperature of ddd chamber
T_{evap}	C	Evaporation temperature
$T_{h,i}$	C	Hot stream inlet temperature
$T_{h,o}$	C	Hot stream outlet temperature
T_i	K	Temperature at inlet to path

T_{lump}	K	Temperature of lumped mass
T_m	K	Mean temperature
T_o	K	Temperature at outlet to path
$\overline{T}_{plate}$	K	Mean temperature of plate
T_{ref}	C	Refrigerated space temperature
T_s	K	Suction temperature
T_{sep}	K	Separation temperature
T_{shell}	K	Shell temperature
T_{sink}	K, °C	Temperature of sink thermal reservoir
T_{source}	K, °C	Temperature of source thermal reservoir
u_{CV}	kJ/kg	Mixture specific internal energy for a given control volume
u_{d1}	kJ/kg	Specific internal energy in d_1 chamber
u_{d2}	kJ/kg	Specific internal energy in d_2 chamber
u_{dd}	kJ/kg	Specific internal energy in dd chamber
u_{ddd}	kJ/kg	Specific internal energy in ddd chamber
u_g	kJ/kg	Gas specific internal energy
u_l	kJ/kg	Liquid specific internal energy
u_m	kJ/kg	Mixture specific internal energy
U	m/s	Velocity
$\overline{U}$	m/s	Mean velocity
U_{CV}	kJ	Total internal energy of a given control volume
UA_{amb}	kW/K	Overall heat transfer conductance with ambient
v	m^3/kg	Specific volume
v_e	m^3/kg	Momentum effective mixture specific volume
$v_{e,up}$	m^3/kg	Effective mixture specific volume at upstream pressure
$v_{e,down}$	m^3/kg	Effective mixture specific volume at downstream pressure
v_g	m^3/kg	Gas specific volume
v_l	m^3/kg	Liquid specific volume
v_m	m^3/kg	Mixture specific volume

V	m^3	Volume
V	m/s	Velocity
V_c	m^3	Volume of compression chamber
$V_{c1,\alpha}$	m^3	Volume of the α-th compression chamber
$V_{c2,\alpha}$	m^3	Volume of the α-th compression chamber
$V_{c2,d}$	m^3	Volume of Nc-th c_2 chamber at discharge angle
$\dot{V}_{comp}$	m^3/h	Displacement rate of compressor
V_{CV}	m^3	Volume of a given control volume
V_{d1}	m^3	Volume of discharge chamber d_1
V_{d2}	m^3	Volume of discharge chamber d_2
V_{dd}	m^3	Volume of discharge chamber dd
V_{ddd}	m^3	Volume of discharge chamber ddd
V_{disp}	m^3	Displacement of compressor
V_i	m^3	Volume enclosed by inner involute
$V_{I,d1}$	m^3	Volume of "I" part of d_1 chamber
$V_{I,sa}$	m^3	Volume of "I" part of sa chamber
$V_{I,s1}$	m^3	Volume of "I" part of s_1 chamber
$V_{Ia,d1}$	m^3	Volume of "Ia" part of d_1 chamber
$V_{Ia,dd}$	m^3	Volume of "Ia" part of dd chamber
$V_{Ia,s1}$	m^3	Volume of "Ia" part of s_1 chamber
$V_{Ib,d1}$	m^3	Volume of "Ib" part of d_1 chamber
$V_{Ib,dd}$	m^3	Volume of "Ib" part of dd chamber
$V_{Ib,s1}$	m^3	Volume of "Ib" part of s_1 chamber
$V_{Ic,d1}$	m^3	Volume of "Ic" part of d_1 chamber
$V_{Ic,s1}$	m^3	Volume of "Ic" part of s_1 chamber
$V_{Id,d1}$	m^3	Volume of "Id" part of d_1 chamber
V_o	m^3	Volume enclosed by outer involute
$V_{Oa,dd}$	m^3	Volume of "Oa" part of dd chamber
$V_{Ob,dd}$	m^3	Volume of "Ob" part of dd chamber

$V_{Oc,dd}$	m^3	Volume of "Oc" part of dd chamber
$V_{O,d1}$	m^3	Volume of "O" part of d_1 chamber
$V_{O,dd}$	m^3	Volume of dd chamber
$V_{O,s1}$	m^3	Volume of "O" part of s_1 chamber
V_{ratio}	-	Volume ratio
V_s	m^3	Volume of suction chamber
V_{s1}	m^3	Volume of s_1 chamber
V_{s2}	m^3	Volume of s_2 chamber
V_{sa}	m^3	Volume of suction area sa
$\dot{W}_{comp}$	kW	Power of compressor
$\dot{W}_{el}$	kW	Electrical power of compressor
$\dot{W}_{exp}$	kW	Power of expander
$\dot{W}_{hyd.exp.}$	kW	Power of hydraulic expander
$\dot{W}_{hyd.exp.,l}$	kW	Electrical power of hydraulic expander from liquid
$\dot{W}_{hyd.exp.,g}$	kW	Electrical power of hydraulic expander from gas
$\dot{W}_i$	kW	Isentropic power of compressor
$\dot{W}_{ML}$	kW	Mechanical losses
$\dot{W}_{rev}$	kW	Reversible power
$\dot{W}_{pump}$	kW	Power of pump
x	-	Vapor quality
x	m	Cartesian coordinate
$x_{\bigcirc}$	m	x-coordinate for crank pin
x_{a1}	m	x-coordinate for center of arc 1
x_{a2}	m	x-coordinate for center of arc 2
$x_{a1,t}$	m	x-coordinate for tangent point of arc 1
$x_{a2,t}$	m	x-coordinate for tangent point of arc 2
x_c	m	x-coordinate for centroid of differential element
x_c	m	x-coordinate for center point of arc
x_{d-dd}	m	x-coordinate for d-dd break line

x_e	m	x-coordinate for ending angle
x_{fi}	m	x-coordinate for fixed scroll inner involute
x_{fis}	m	x-coordinate for fixed inner starting angle
x_{fo}	m	x-coordinate for fixed scroll outer involute
x_{fos}	m	x-coordinate for fixed outer starting angle
x_g	-	Gas mass fraction
$x_{g,s}$	-	Gas mass fraction solved in oil
$x_{g,v}$	-	Gas mass fraction in vapor phase
x_k	m	x-coordinate for conjugate point
x_l	-	Liquid mass fraction
$x_{l,d1}$	kg	Oil mass fraction in d_1 chamber
$x_{l,d2}$	kg	Oil mass fraction in d_2 chamber
$x_{l,dd}$	kg	Oil mass fraction in dd chamber
$x_{l,ddd}$	kg	Oil mass fraction in ddd chamber
$x_{l,f}$	-	Upstream oil mass fraction of flow path
$x_{l,opt}$	-	Optimal oil mass fraction
$x_{l,s}$	-	Liquid mass fraction in solution
x_{oi}	m	Coordinates for orbiting inner involute
x_{oo}	m	Coordinates for orbiting outer involute
x_{oos}	m	x-coordinate for orbiting outer starting angle
x_p	m	x-coordinate for point of integration
$x_{s,sa}$	-	x-coordinate of s_1 break angle
X_d	-	Area correction factor
X_{sep}	-	Mass fraction of oil in solution
y	m	Cartesian coordinate
$y_{\bigcirc}$	m	y-coordinate for crank pin
$\mathbf{y}$	-	Vector of properties
y_{a1}	m	y-coordinate for center of arc 1
y_{a2}	m	y-coordinate for center of arc 2

$y_{a1,t}$	m	y-coordinate for tangent point of arc 1
$y_{a2,t}$	m	y-coordinate for tangent point of arc 2
y_c	m	y-coordinate for center point of arc
y_c	m	y-coordinate for centroid of differential element
y_{d-dd}	m	y-coordinate for d-dd break line
y_e	m	y-coordinate for ending angle
y_{fi}	m	y-coordinate for fixed scroll inner involute
y_{fis}	m	y-coordinate for fixed inner starting angle
y_{fo}	m	y-coordinate for fixed scroll outer involute
y_{fos}	m	y-coordinate for fixed outer starting angle
y_g	-	Molar fraction of gas
y_k	m	y-coordinate for conjugate point
y_{oi}	m	y-coordinates for orbiting inner involute
y_{oo}	m	y-coordinates for orbiting outer involute
y_{oos}	m	y-coordinate for orbiting outer starting angle
y_p	m	y-coordinate for point of integration
$y_{s,sa}$	m	y-coordinate of s_1 break angle
α	-	Index of compression chamber
α	-	Void fraction
α	rad	Discharge region angle
β	rad	Discharge region angle
χ_{old}	-	Old state variable at wrapping
χ_{new}	-	New state variable at wrapping
δ	m	Leakage gap width
δ_{flank}	m	Flank leakage gap width
δ_{radial}	m	Radial leakage gap width
Δh_{Reg}	kJ/kg	Specific enthalpy change in regenerator
$\Delta h_{Reg,max}$	kJ/kg	Maximum specific enthalpy change in regenerator
Δp	kPa	Pressure difference

ΔT_{pinch}	K	Pinch temperature difference
ΔT_{sh}	K	Superheat temperature difference
ΔT_{sc}	K	Subcooling temperature difference
$\Delta\theta$	rad	Step size
$\Delta\theta_{old}$	rad	Old step size
$\Delta\theta_{new}$	rad	New step size
η_a	-	Adiabatic efficiency of compressor
η_c	-	Adiabatic efficiency of compressor
η_{comp}	-	Adiabatic efficiency of compressor
η_{guess}	-	Guess for adiabatic efficiency of compressor
$\eta_{hyd-exp,l}$	-	Adiabatic efficiency of hydraulic expander for liquid
$\eta_{hyd-exp,g}$	-	Adiabatic efficiency of hydraulic expander for gas
$\eta_{II,AC}$	-	Second Law efficiency in cooling mode
$\eta_{II,HP}$	-	Second Law efficiency in heating mode
$\eta_{oi,comp}$	-	Overall isentropic efficiency of compressor
$\eta_{oi,exp}$	-	Overall isentropic efficiency of expander
$\eta_{oi,hyd.exp.}$	-	Overall isentropic efficiency of hydraulic expander
$\eta_{oi,pump}$	-	Overall isentropic efficiency of hydraulic pump
η_v	-	Volumetric efficiency
ε_{Reg}	-	Effectiveness of regenerator
ϵ	-	Vector of errors per step
ϵ_{HX}	-	Heat exchanger effectiveness
ε	-	Convergence criterion
$\varepsilon_{allowed}$	-	Maximum error allowed per step
ε_{max}	-	Maximum error for given step
γ	radian	Circular sector angle
μ_g	Pa-s	Gas viscosity
μ_l	Pa-s	Liquid viscosity
μ_m	Pa-s	Mixture viscosity

ω	rad/s	Rotational speed of compressor
ϕ	rad	Involute angle
ϕ_0	rad	Initial involute angle
ϕ_1	rad	First involute
ϕ_2	rad	Second involute angle
ϕ_{d-dd}	rad	Involute angle separating d and dd chambers
ϕ_{i0}	rad	Inner involute initial angle
ϕ_{is}	rad	Inner involute starting angle
ϕ^*_{is}	rad	Effective inner involute starting angle
ϕ_{ie}	rad	Inner involute ending angle
ϕ^*_{ie}	rad	Effective inner involute ending angle
ϕ_e	rad	Ending involute angle
ϕ_{fi0}	rad	Fixed scroll inner involute initial angle
ϕ_{fis}	rad	Fixed scroll inner involute starting angle
ϕ_{fie}	rad	Fixed scroll inner involute ending angle
ϕ_{fo0}	rad	Fixed scroll outer involute initial angle
ϕ_{fos}	rad	Fixed scroll outer involute starting angle
ϕ_{foe}	rad	Fixed scroll outer involute ending angle
$\phi_{k,fi}$	rad	Conjugate angles for fixed inner involute
$\phi_{k,fo}$	rad	Conjugate angles for fixed outer involute
$\phi_{k,oi}$	rad	Conjugate angles for orbiting inner involute
$\phi_{k,oo}$	rad	Conjugate angles for orbiting outer involute
ϕ_m	rad	Mean involute angle
ϕ_{max}	rad	Maximum involute angle
ϕ_{min}	rad	Minimum involute angle
ϕ_{o0}	rad	Outer involute initial angle
ϕ_{os}	rad	Outer involute starting angle
ϕ_{oe}	rad	Outer involute ending angle
ϕ_{oi0}	rad	Orbiting Scroll inner involute initial angle

ϕ_{ois}	rad	Orbiting Scroll inner involute starting angle
ϕ_{oie}	rad	Orbiting Scroll inner involute ending angle
ϕ_{oo0}	rad	Orbiting Scroll outer involute initial angle
ϕ_{oos}	rad	Orbiting Scroll outer involute starting angle
ϕ_{ooe}	rad	Orbiting Scroll outer involute ending angle
ϕ_s	rad	Starting involute angle
$\phi_{s,sa}$	rad	Break angle for suction chamber
ψ	-	Entrainment factor
Ψ	-	Compression chamber centroid term
ρ	kg/m^3	Density
$\overline{\rho}$	kg/m^3	Mean density
ρ_{CV}	kg/m^3	Density of a given control volume
ρ_g	kg/m^3	Gas density
ρ_l	kg/m^3	Liquid density
ρ_m	kg/m^3	Mixture density
$\rho_{m,in}$	kg/m^3	Inlet mixture density
ρ_{up}	kg/m^3	Upstream density
σ	-	Ratio of upstream to downstream area
τ	kN-m	Shaft torque
$\overline{\tau}$	kN-m	Mean shaft torque over one rotation
θ	rad	Crank angle of shaft
θ_d	rad	Discharge angle
θ_m	rad	Offset crank angle
θ_Δ	rad	Offset for crank angle

ABSTRACT

Bell, Ian Hadley Ph.D, Purdue University, May 2011. Theoretical and Experimental Analysis of Liquid Flooded Compression in Scroll Compressors. Major Professors: Eckhard A. Groll, School of Mechanical Engineering and James E. Braun, School of Mechanical Engineering.

Adding liquid to the working fluid in scroll compressors can allow for a working process that approaches isothermal compression. When liquid flooding and regeneration is applied to refrigeration and heat pump systems, simple cycle modeling predicts that for systems that operate at very large temperature lifts, the increase in system coefficient of performance can be greater than 50%. In order to better understand the liquid-flooded working process, a detailed scroll compressor model has been developed which comprises a geometric model and an overall compressor model. The geometric model includes numerically validated analytic solutions for all geometric parameters, including force terms, for constant wall thickness scroll wraps that can have multiple pairs of compression chambers. The overall model includes a frictionally-corrected isentropic nozzle leakage model, adaptive Runge-Kutta solver for the system of differential equations, and numerically efficient thermodynamic and transport property routines. The compressor model has been validated against testing conducted on the Liquid-Flooded Ericsson Cycle for oil mass fractions as high as 92% oil by mass with error in predictions of shaft power and mass flow less than 3%. Optimization of the compressor performance with flooding for several applications is carried out, and with optimization, overall isentropic efficiencies over 75% are predicted for configurations with large amounts of oil flooding. Further testing on a refrigerant R410A vapor injected compressor with oil injection has shown that the performance with oil injection improves monotonically with the oil injection rate as long as the oil is cooled prior to injection.

CHAPTER 1. INTRODUCTION, MOTIVATION, AND OBJECTIVES

1.1 Background

In the late 1800s natural refrigerants such as CO_2 were the norm, largely because artificial working fluids had not yet been introduced. It was only with the advent of R12 in the 1930s that the transition was made away from natural refrigerants. At their introduction, the new refrigerants seemed like a panacea; good thermodynamic performance with no deleterious impacts on the environment. These new artificial refrigerants quickly gained massive adoption, primarily as working fluids in cooling systems.

During the early 1970s, two pairs of researchers (Cicerone and Stolarski from the University of Michigan and Rowland and Molina of the University of California, Irvine) were investigating the impact of chlorinated fluids on the ozone layer. By 1976, the National Research Council of the USA was of the opinion that chlorinated refrigerants could catalyze the destruction of the ozone (1976). After the chlorofluorocarbon refrigerants had been conclusively shown to be causing damage to the ozone layer, different directions were needed in the field of refrigeration working fluids. While there is still a large interest in developing new refrigerants that can perform cooling without environmental destruction, another means to solve this problem is to use working fluids that are naturally occurring and safe.

In order to effectively use these natural working fluids, it is critical that they achieve levels of efficiency on par with those of existing working fluids if they are to be commercially competitive. Table 1.1 shows the properties and efficiency of a range of working fluids. For applications where the heat rejection temperature is close to or greater than the critical temperature of CO_2 (31.0°C), carbon dioxide needs some

optimization to be competitive with other refrigerants. In spite of its relatively poor efficiency at high ambient temperatures, carbon dioxide (CO_2) is non-flammable and non-toxic, and commonly existing in the environment. This is why there is a large interest in the optimization of carbon dioxide cycle efficiency, through the use of a wide range of techniques, predominantly focused on decreasing the exergetic losses in the throttling process and the heat transfer irreversibilities in the compressor discharge superheat horn. One possible means of improving the efficiency of carbon dioxide refrigerating cycles is through the use of liquid flooding to achieve a more isothermal compression process.

Table 1.1 Properties of selected refrigerants (Lorentzen, 1995).

Refrigerant	R-22	R134a	R-410A	R-404A	R-717	R-290	R-744
Name	Freon	-	Puron	-	ammonia	propane	CO_2
Natural	No	No	No	No	Yes	Yes	Yes
ODP[a]	0.05	0	0	0	0	0	0
GWP[b]	1500	1200	1730	3900	0	20	1
COP[c]	4.65	4.60	4.41	4.21	4.84	4.74	2.96
Flammable	No	No	No	No	Yes	Yes	No

[a]Ozone depletion potential (normalized so that ODP of R11 is 1.0)

[b]Global warming potential (normalized so that the GWP of CO_2 is 1.0)

[c]Cycles operating between 258K evaporating and 303K condensing, 0K subcooling, 0K superheating (Pearson, 2005)

1.2 Motivation

Hugenroth (2006) proposed to flood a gas refrigeration cycle with oil in order to approach isothermal compression, with the ultimate goal of developing a system to be used in beverage coolers with a COP of 1.25. This Liquid Flooded Ericsson Cycle

(LFEC) was a novel application of gas refrigeration, but in the course of Hugenroth's study several new lines of inquiry were opened.

One of the questions raised by Hugenroth's work was related to the modeling of the flooded compression process. No detailed analysis of the compression process had been carried out, and as with other gas refrigeration cycles, the system is very sensitive to the efficiency of the rotating machinery. To achieve the COP of 1.25, compressor adiabatic efficiency on the order of 87 % was required. To achieve this challenging target it is necessary to have a thorough understanding of the losses produced in the compressor and develop means of eliminating them. Therefore detailed modeling of the scroll compressor is one of the main goals of this work.

Hugenroth began investigating flooded compression in vapor compression applications (Hugenroth et al., 2006), and he found that it was possible to improve the overall system efficiency by flooding the compressor with oil. Further analysis of vapor compression systems with flooding has been carried out, the results of which are presented in this study. From this analysis, carbon dioxide seems to be a promising avenue of research with flooding, and will be further pursued. Other refrigerants also demonstrate significant improvements in efficiency with flooding and regeneration.

1.3 Objective

The overall objective of the work presented here is to develop a fundamental understanding of liquid flooding and its impact on system performance. In particular the emphasis is on understanding the compression process in scroll compressors since the remaining processes in the refrigeration system are better understood. To that end, a mechanistic, physics-based, model has been developed in order to understand the liquid-flooded scroll compression process. This model includes comprehensive analytic solutions for the scroll compressor geometry, mixture properties, and corrected leakage flow rates. There are currently no other models in open literature that include all these effects.

A further objective is to optimize a scroll compressor for liquid flooding based on the model developed. Scroll compressors are optimized for applications using nitrogen, CO_2 and refrigerant R410A as working fluids. Finally, tests are carried out on a hermetic scroll compressor in order to better understand the working process of the liquid-flooded compression process.

1.4 Overview

Chapter 2 presents a thorough study of the state-of-the-art literature in flooded compression in screw and scroll compressors as well as system-wide analysis for flooded compression. Chapter 3 provides simplified system modeling for liquid-flooded vapor compression cycles which motivates the further study of the flooded compression process in scroll compressors. Chapter 4 develops the geometry of the scroll compressor which is needed in the thermodynamic model of the scroll compressor presented in Chapter 5. Chapter 6 describes the experimental testing carried out on the Liquid Flooded Ericsson Cycle and provides experimental validation of the modeling presented in the prior chapter. Chapter 7 presents the results of the optimization of scroll compressors for liquid flooding. Chapter 8 presents experimental data obtained from the testing of a R410A scroll compressor with oil flooding. Finally Chapter 9 summarizes all the work presented here.

CHAPTER 2. LITERATURE SURVEY

2.1 Flooded Compression

This section summarizes the state-of-the-art in flooded compression analysis and modeling - both the influence of oil flooding on particular compressor types with oil injection, as well as the cycle performance with oil flooding. In addition, literature relating to refrigerant-oil solubility is introduced.

2.1.1 Screw Compressors

A number of researchers have investigated injecting significant amounts of oil into the compression chamber of screw compressors. The motivation for oil injection in screw comprssors is that oil should be able to seal the leakage gaps between each of the rotors and between the rotors and housing. In addition the oil can help to control the outlet temperature of the compressor by absorbing some of the heat of compression of the refrigerant as it goes through the compression. As screw compressors generally see high pressure ratios that are typically above those of centrifugal compressors, the discharge temperatures can be quite high.

Bein and Hamilton (1982) modeled the flooding process in screw compressors. Their leakage modeling was based on the flow of oil through orifices with correction coefficients where they assumed that the leakage was entirely oil due to the centrifugal flinging of the oil to the leakage gaps. The discharge port was modeled as compressible flow of perfect gas. Limited modeling results were presented, the primary conclusion being that the suction air preheat and the discharge port pressure drop were the dominant parameters causing a reduction in the volumetric efficiency.

Singh and Patel (1984) investigated twin-screw compressors with oil-flooding. They derived differential relations for the temperature, pressure and density of the gas and mass and temperature of the oil and then integrated these differential relations to obtain the total power. They used a correction coefficient to account for the leakage blockage by the injected oil but did not provide a value for this coefficient.

Blaise and Dutto (1988) experimentally investigated the performance of a single-screw refrigeration compressor with oil flooding. They found that the injection of oil into the compression pocket can help to seal the rotor leakages, and found that the volumetric efficiency of the compressor increased monotonically with the amount of oil injected for sealing. The isentropic efficiency approached an asymptote for higher rates of sealing oil injection.

Stosic et al. (1988) modeled flooded compression in screw compressors. They used continuity and conservation of energy to calculate the derivatives of temperature, pressure and oil mass fraction. Their model also included treatment of the oil-gas thermal exchange by assuming that the oil was liquid droplets and using a liquid droplet heat transfer correlation. From experimental measurements they found that the volumetric efficiency increased monotonically with the oil/gas mass flow ratio. The volumetric efficiency was also seen to increase monotonically with an increase in the viscosity of the oil. The same trends were seen for the isothermal efficiency.

Wu and Jin (1988) carried out similar modeling of the oil-flooded screw compressor. They treated the suction flow as the flow of a compressible perfect gas, the discharge flow was treated as flow through an orifice, and the leakages were treated as homogenous oil-gas flow, oil flow between parallel plates, and separated flow. They also found that the volumetric and adiabatic efficiencies increase monotonically with the oil flow rate over the range of oil flow rates tested.

Stosic et al. (1990) experimentally measured the performance of an oil-flooded air screw compressor for which they found that there was an optimal oil flow rate which minimized the outlet temperature of the compressor. The modeling presented by Stosic et al. (1988) was found to predict the efficiency of the machine quite well.

An optimal amount of liquid flooding was also found which minimized the specific power.

Tang and Fleming (1992) also constructed a mathematical model for flooded compression, similar in formulation to other authors (Stosic et al., 1988; Wu and Jin, 1988). The differences are that Tang and Fleming considered compressibility through the leakages, assumed no heat transfer between fluid and rotors, and considered flashing of refrigerant out of solution in oil. Qualitatively good agreement between experimental and modeling results was found. No variation of the oil flow rate was carried out in the experimental procedure.

Fujiwara and Osada (1995) also modeled flooded screw compressors. Their model treated all flows besides the lobe tip clearance as isentropic flow through a nozzle with modified adiabatic exponents and gas constants. The leakages are treated as being all liquid and incompressible viscous flow through channels is used to calculate the leakage flow rate. Flow coefficient values are also used to correct the flow rates, with values between 0.4 and 1.0. A heat transfer correlation was derived for the heat transfer in the suction, compression, and discharge processes. Model predictions agreed well with experimental data.

Wu et al. (2004) also carried out modeling of flooded screw compressors. They treated the leakage flow as separated slip flow with entrainment, but it is believed that there is a typo in their expression for the slip ratio. With the typo corrected, their slip ratio term becomes that of Chisholm (1983). Wu also considered the heat transfer process for the injected oil. The modeling did not accurately capture the discharge gas pulsations but otherwise was successful at predicting the shape of the p-V curve.

Li and Jin (2004) carried out optimization of a flooded screw compressor. They found that there is an optimal flow rate of oil which depends on the injected oil temperature. This optimal oil flow rate minimizes the volumetric specific work.

2.1.2 Scroll Compressors

The first detailed compressor modeling of oil-flooded scroll compressors was carried out by Li et al. (1992). Their primary interest was to use the oil to decrease clearances as well as improve the lubrication in the compressor. They found that the injection of oil (though the quantity is not presented) could yield a decrease in specific power of 9.41%, an increase in volumetric efficiency of 4.3%, and reduce the discharge temperature by 30%. The discussion of their model is quite limited, but they used conservation of mass and energy to arrive at a set of differential equations. In addition, they used a frictional annulus flow model for the radial leakages and flow between conformal cylinders for the flank leakages, though they do not state whether they had assumed oil or gas to flow through their leakage paths.

Qu and Tramschek (1996) investigated the oil flooding of two-stage air compressors, including experimental and modeling work. Their work found there was a monotonic increase in air flow rate with oil injection. The oil injection rate was up to approximately 1.0 % by volume.

Sakuda et al. (2001) investigated flooded compression in scroll compressors. They modified the oil control valve for the compressor to reduce the oil injection rate, and found that the efficiency increased for decreasing oil mass fractions.

Hiwata et al. (2002) presented experimental results on flooded compression of a 4 cm^3 displacement CO_2 scroll compressor. Depending on the state point considered, the optimal oil flow rate was between 6% and 14.8%. Discharge pressure and temperature oil was directly injected into the suction chambers. They proposed that the optimal oil flooding rate was a tradeoff between leakage losses at low oil injection rates and suction preheating at high oil flow rates. Limited information is presented about the compressor investigated.

Sawai et al. (2009) continued the work of Sakuda et al. (2001) in the oil injection to scroll compressors. They also found that the coefficient of performance of the system improved as the oil circulation rate was reduced.

Toublanc (2009) investigated injecting oil into the compression process of a transcritical CO_2 compressor, and developed detailed mechanistic models for the oil injection process. Experimental validation of the oil injection process is included.

2.1.3 Spool Compressors

Spool compressors are novel compressors somewhat similar in design to a rotary compressor. These compressors can accept liquid flooding. Kemp et al. (2010) have investigated a liquid-flooded spool compressor with oil injection rates up to 30%, and they found a monotonic increase in overall isentropic efficiency as well as volumetric efficiency with increasing oil flow rates.

2.1.4 Cycle and System Analysis

With regards to flooded compression, there has historically been an interest in using the vaporization of water to precool the gas entering into gas turbines in order to decrease its temperature and increase its density. This is the technique investigated by Zheng et al. (2003) as well as White and Meacock (2004). They found that the evaporation of water can be used to cool the inlet gas, but the results are not as good as predicted from simplified modeling.

Hugenroth (2006; 2007) carried out cycle modeling for the Liquid Flooded Ericsson Cycle (LFEC) which provides the basis for the work presented here. From the results of the system tests, it was clear that liquid flooding could provide benefits to system performance if some of the parasitic losses like pressure drops could be overcome. From simplified modeling Hugenroth found that there was an optimal oil-flooding rate at a capacitance ratio of 7, but experimental tests did not bear out this optimal value. The compressor adiabatic efficiency decreased monotonically with the amount of oil injection. This was suggested to be be due to compressor discharge pressure drops.

Hugenroth (2006) also investigated the use of oil-flooding in vapor compression systems. Based on simplified system modeling which was experimentally validated for dry operation Hugenroth found that the addition of oil could result in an increase in system COP up to 13% in heating mode and 9% in cooling mode. This simplified modeling did not take losses like pressure drops in the compressor or system components into account.

There has been commercial interest in flooded compression, and the patent of Ignatiev (2008) covers the technology of liquid flooding applied to vapor compression systems. This patent includes the addition of regeneration.

Lottin et al. (2003a; 2003b) investigated the impacts on cycle performance of circulating varied amounts of oil through the entire cycle. They find that the COP decreases sharply with increased oil circulation, and by an oil circulation rate of 1% by mass, the COP decreases between 3 and 9 percent depending on the system configuration and operating state point. This work also includes simple models for estimating the physical properties of oils and oil-refrigerant blends.

2.1.5 Solubility

In order to fully characterize the mixture of refrigerant and liquid during liquid-flooded cycle operation, it is necessary to understand the solubility of the refrigerant in the flooding liquid.

The working fluid for the Liquid-Flooded Ericsson Cycle introduced by Hugenroth (2006) and most of the scroll compressor modeling results presented here is nitrogen gas. Thus, it is necessary to have models and data for the solubility of nitrogen in a range of working fluids. Logvinyuk (1970) presents data for the solubility of nitrogen in a range of petroleum products. Totten (2003) presents experimental data for the solubility of nitrogen in both PAG and PAO oil. Lawrie (1928) presents experimental data for the solubility of nitrogen and carbon dioxide in glycerol. Battino et al. (1984) summarize all the solubility data from literature for the solubility of nitrogen in fluids.

Both organic and inorganic fluids are included. The primary conclusion to be drawn from this body of data is that at reasonable pressures, the solubility of nitrogen in the liquid is quite low and can be easily neglected.

If the cycle operates with CO_2 as the working fluid, the impact of solubility is greater due to both the tendency of CO_2 to solve in other fluids (partly why supercritical CO_2 is finding favor as an environmentally friendly cleaning agent), as well as the high working pressures experienced in transcritical CO_2 systems. Seeton (2000) presents solubility and viscosity data for mixtures of CO_2 with different types of oils: polyol-ester (POE), poly-alkylene glycol (PAG), poly-α-olefin (PAO), and alkyl-benzene (AB). In a further study, Seeton (2006) presents thermophysical data for mixtures of CO_2 and POE32 oil. Hauk (2001; 2000) presents solubility data for CO_2 with PAO, POE, and PAG oils over a range of conditions. Hauk also demonstrates some complex phase distribution behaviors like phase inversion and vapor-liquid-liquid equilibrium. Fahl (2002) provides further data for CO_2 and a wide range of oils. Garcia (2008) fit equations of state to the experimental data of Hauk for the solubility of CO_2 in oils as well as a number of other researchers. Fandiño (2008) investigated the solubility of CO_2 in pentaerythritol ester oils. Water is an excellent flooding agent for the liquid-flooded system as will be shown later, and Duan et al. (2003) present data for the solubility of CO_2 in water over a range of conditions. In general the solubility of CO_2 in water is much lower than in refrigeration oils at the same temperature and pressure.

There is also interest in the use of R410A or other hydrofluorocarbon-blends in liquid-flooded cycle applications, for which there is a relative paucity of solubility data available. Martz (1994; 1996) modeled mixtures based on mixtures of R12, R22, R134a, and R125 (R125 is a constituent of R410A). Burton (1997; 1999) measured and modeled the solubility of R32 in POE oil and predicted the solubility of R410A. Both Burton (1997) and Martz (1994) provide the source code used to evaluate the equations of state. The VDI Wärmeatlas (Pfenning, 2010) provides a flow-chart that explains how to use the equations of state.

In order to be able to predict the phase behavior of mixtures of refrigerants and oil, it is necessary to develop equations of states for the mixtures. Yokozeki (2001; 2005; 2007) and Youbi-Idrissi (2008; 2003; 2004) have modeled mixtures of oils and refrigerants, as well as some other researchers (Mermond et al., 1999; Teodorescu et al., 2003). Yokozeki has also proposed a universal oil *UniOIL* for modeling purposes (2001) as well as for investigating the time-dependence of solubility (2002). Youbi-Idrissi (2008) presents a thorough review of all the cycle and component impacts caused by oil circulation.

Conde (1996) also investigated the thermodynamics of the mixture of oil and refrigerant, from which it is suggested that the heat of mixing can be neglected for oil-refrigerant mixtures since it is much smaller than other changes in enthalpy.

2.1.6 Compressed Air Energy Storage

Currently there is a great deal of interest in alternative, green energy sources that provide intermittent power. For example, when the wind is blowing wind turbines can generate large quantities of power, but when the wind stops they cannot generate any power. Various means have been proposed to shift the load and provide a more even power generation profile, and one approach is to use wind power to compress air and then recover the compression power at a later time. This is the method proposed by Wang (2006), and is a promising application. Flooded compression would be ideally suited to this application since large pressure ratios are to be expected and liquid flooding can help to reduce the discharge temperature. Additionally the high pressure, high-temperature liquid could be separated off and stored and then used to preheat the gas during the expansion phase.

2.2 Modeling Of Scroll Compressors

The scroll compressor was first proposed by Creux (1905). In the subsequent 75 years, little progress was made towards feasible scroll compressor designs. After the

advent of accurate numerically controlled tooling, it was then possible to machine the scroll involute profiles in an accurate and reliable manner. In the last twenty-five years, scroll compressors have obtained a dominant market position in the fields of air conditioning and refrigeration, and as a result there has been a great deal of academic and industrial work carried out on the modeling of scroll compressors. Figure 2.1 shows the number of papers dealing with scroll machines at the Purdue University bi-annual conference on compressor engineering. Only after the mid-1980s are there a significant number of academic and industrial papers published on scroll compressors.

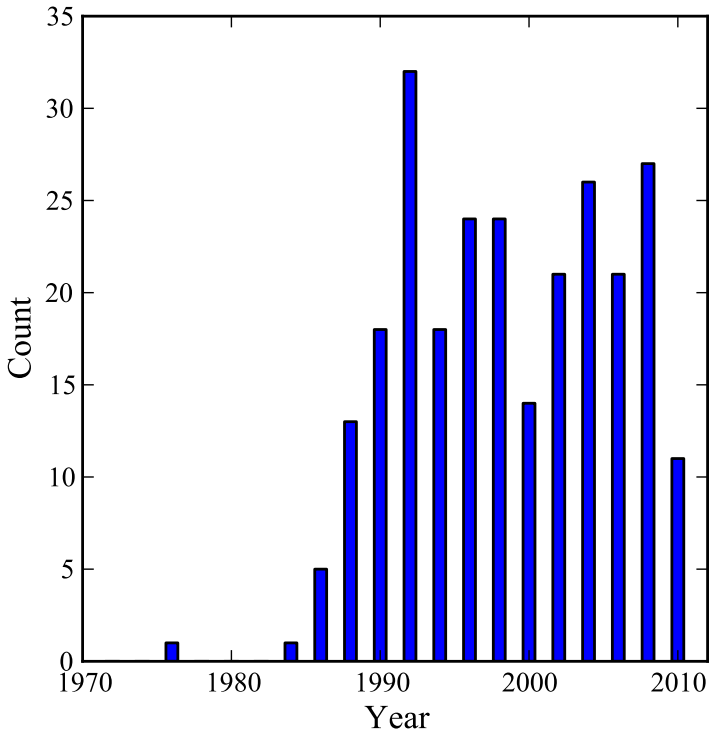


Figure 2.1. Papers of the Purdue University International Compressor Engineering Conference related to scroll compressors and scroll expanders.

2.2.1 Geometry

A significant amount of literature exists which deals with the geometric modeling of scroll compressors. Morishita (1984) produced one of the first complete analyses of the scroll compressor including geometry and dynamics. Yanagisawa (1990) constructed a full geometric model based on the foundation set by Morishita. More treatment of the scroll geometry are available from Halm (1997). One of the major shortcomings of Halm's model is that the derivation is based on particular values for the initial angles of involute. Wang (2005) noted this problem and carried out derivations for the scroll chamber geometries with arbitrary initial involute angles. With this level of generality it is then possible to treat a wide range of compressor geometries with constant wrap thickness.

Other frameworks for scroll wrap analysis have been proposed, including curves derived by enforcing conjugacy in the most general way (Bush and Beagle, 1992). Furthermore, it can be of use to allow for variable scroll wrap thickness, and the analysis of Gravesen (2001) or Bush (1994) can be used to analyze the geometry for variable wall thickness scroll wraps. Blunier (2006) also provides an analytic solution for the scroll chamber geometry based on the model of Gravesen for constant wall thickness. Other types of scroll wrap geometry have been proposed, including the involute of a square (Wang, 1992).

The discharge chamber has complex geometry, and in addition to the analysis provided below, Lee and Wu have carried out derivations for the perfect-meshing-profile (PMP) for scroll compressors (Lee and Wu, 1995). Blunier (2009) has also provided analytic models for the volume of the discharge chamber geometry for single arc and two-arc discharge chamber geometry.

2.2.2 Mass Flow Modeling

Leakage is one of the most challenging elements to model for scroll compressors, and this is particularly true in the case that the compressor is flooded with liquid.

Restricted to only single-phase gas flow, there are still a number of models available. Some models consider friction, some treat the flow as isentropic compressible flow through a nozzle; others consider the flow as through an orifice. Accurate modeling of the leakage flow is critical to gaining a complete understanding of the sensitivity of compressor performance to various design parameters. Once the library of potential flow models is opened up to two-phase flow, an accurate selection of flow model becomes even more difficult.

Single-Phase Models

The typical baseline model for leakage in scroll compressors is the compressible flow of a perfect gas through an isentropic, converging-diverging nozzle. This model allows for choking when the Mach number reaches 1 at the throat. This model is typically used with empirical correction factors to compensate for static pressure losses in the flow path due to frictional effects. This is the model of choice for a number of authors, including Margolis (1992), Puff (1992), Youn (2000), Lee (2002), Chen (2002), among many others. The primary motivating factor for the use of this model is its simplistic form as only one area, the throat area, is required. In addition the mass flow is explicitly obtained from the compressible mass flow expression, adding little computational overhead if implemented into a detailed model. Typically this model is applied to both flank and radial leakages. One of the challenges is the determination of the discharge coefficient, and limited experimental data is available, but Cho et al. (2000) have investigated this problem and found a discharge coefficient of 0.1 fit their choked flow data well, which suggests that the isentropic flow model does not do a very good job of capturing the actual mass flow rate since such a large correction is required.

One of the major shortcomings of the isentropic compressible single-phase flow model is that it does not take friction into account. The leakage gaps are relatively long relative to the leakage gap widths. Therefore, friction can be expected to play a

significant role in the flow through the leakages. Frictional flow models can be further categorized based on their treatment of compressibility; some models treat the fluid as incompressible, others as compressible. For incompressible flow, the pressure drop over the leakage path can be calculated from incompressible pipe flow relations, as suggested by Ishii (1996a). Very good agreement is found with experimental measurements carried out on a specialized test stand built to test leakage characteristics. Yuan (1992) and Fan (1994) extended the incompressible flow with friction model to account for the inertial terms in the Navier-Stokes equations which are neglected in the pipe flow analysis but end up at an expression which needs to be integrated over the flow path. They found that their model provides results that are superior to that of either isentropic compressible nozzle flow or to pipe flow. Kang (2002) found that using compressible adiabatic flow with friction (Fanno Flow) gave a good match to the predictions of FLUENT results, and was superior to the use of isentropic compressible nozzle flow. Suefuji (1992) also found good results by using Fanno flow through the leakages.

Li presents a model for the radial leakage flow based on radial outflow through a cylindrical section (Li et al., 1992), but is missing units, while Yanagisawa (1985b) presents the same model with the necessary description and units. Tseng (2006) also uses the same model.

Beyond the simplified models, there are a number of hybrid models that select elements from several models. Yanagisawa (1985a) and Tojo (1986) used a combined converging isentropic nozzle/compressible frictional flow section to model the flow through the leakages. Afjei (1992) used superposition methods to calculate the volumetric flow through the leakages as a sum of the rolling, dragging, pressure driven, and flashing components.

Further extending the hybrid methods, some researchers have used 2-D solutions of the Navier-Stokes equations, as in Huang(1994), but it was found that the error between 1-D and 2-D solutions was less than 10% for the gap widths typical of scroll compressors.

The works of Oku (2006) and Ishii (2008) are focused on the use of CO_2 as a working fluid. Ishii notes that the Mach number of the flow through the leakages is typically below 0.3 and thus, compressibility effects can be neglected. As a result, using an experimental test rig, they calculated the frictional coefficients for CO_2 flowing through the flank and radial leakages using a blow-down test stand. In Oku (2006), results are tabulated for CO_2 dry and with some amount of oil. The study of Ishii et al. (2008) focused on the impact of the roughness of the material forming the leakage path, and they found that the friction goes up as the surface roughness increases.

Multi-Phase Models

The addition of oil through the leakages results in fluid effects which are difficult to handle. Even after decades of diligent research, much is still not understood about two-phase flow, even for simple situations, and the flow in the compressor is anything but. Much of the knowledge about two-phase leakage in scroll compressors is derived from work on flooded screw compressors, but little fundamental research has been carried out on the physics of a two-phase mixture flowing through the leakage gaps of screw compressors. As the leakage in screw compressors is tightly coupled to the rest of the compression process, and dependent on the physical gap widths (typically not known), validation that the proper flow model has been selected is very difficult.

In prior literature on oil-flooded screw compressors, authors have treated the various leakage paths as incompressible oil leakage (Bein and Hamilton, 1982; Fujiwara and Osada, 1995; Wu and Jin, 1988), refrigerant flowing through an orifice with a correction coefficient (Singh and Patel, 1984), homogeneous flow of an oil-refrigerant mixture (Wu and Jin, 1988), separated flow (Wu and Jin, 1988), isenthalpic throttling of gas (Tang and Fleming, 1992), isentropic flow through a nozzle with modified adiabatic exponents and gas constants (Fujiwara and Osada, 1995), or separated flow with entrainment (Wu et al., 2004). In essence, the large variety in mass flow mod-

els employed suggests no consensus on how to properly handle the flow of oil and refrigerant.

Li and Wang (2000) developed a model for the flow of oil and gas through the leakage gaps based on a momentum balance of the two phases. They proposed that there is a gap width below which there is no flow of gas, and only oil flows.

Yanagisawa (1985a; 1985b; 1985c) investigated the modeling of flow leakage for rotary compressors. Rotary compressors have leakage flow paths similar in geometry to that of the scroll compressor. Experimental validation was provided, which is quite unique among leakage flow models. Ishii (1996a; 2008) also provided experimental validation, but his work is more difficult to use as blow-down tests were used, and several questionable assumptions were required in order to calculate the mass flow rate.

There are currently no papers which deal with the leakage modeling of scroll compressors which are oil-flooded, as this is an open field of research. Itoh et al. (1990) did present some results for oil adhesion for flooded application.

Xin et al. (2010) experimentally investigated two-phase flow through the gaps of screw compressors. They found that adding 0.5% oil by volume results in a 77% reduction in the leakage flow rate.

Leakage gap widths

One of the most important aspects of the mass flow modeling is determining the gap widths between the two scrolls. When the compressor is running, the scroll wraps will deform differentially due to mechanical loading from the pressure applied to the scroll wraps as well as thermal expansion. For instance, from the study of Lin, the maximum deflection of the fixed scroll wrap was found to be 24.50 μm and 29.41 μm for the orbiting scroll (Lin et al., 2005). This study took into account the thermal expansion of the scrolls.

Thus, an estimation of the instantaneous gap width for the compressor is quite challenging; to do so requires accounting for the effects of scroll deformation, machining tolerances, compliant mechanisms if installed, thermal expansion, wear, tip seals, overturning moments, etc. Thus, most researchers have assumed all flank gap widths to be fixed, or at the most, constant and a function of the pressure ratio. A summary of the gap widths from literature is in Table 2.1

Table 2.1 Survey of scroll compressor gap widths from literature.

Reference	δ_{flank} $[\mu m]$	δ_{radial} $[\mu m]$
Yang (2008)	49-75	
Lee (2002)	15	10
Suefuji (1992)	5-15	
Schein (2001)	10-30	
Ishii (1996b)	10	
Halm (1997)	0-40	0-5
Youn (2000)	-	50
Lemort (2008)	70	0
Ishii (2008)	6	3

The leakage gap widths from Halm (1997) appear to have a typographical error in the constant multiplying the pressure ratio term[1], the correct forms are believed to be

$$\delta_{flank} = -9.61538\text{x}10^{-6}\left(\frac{p_d}{p_s} - 1.67\right) + 20\text{x}10^{-6} \tag{2.1}$$

$$\delta_{radial} = 1.098\text{x}10^{-6}\left(\frac{p_d}{p_s} - 1.67\right) + 1\text{x}10^{-6}. \tag{2.2}$$

The pressure ratios of Chen (2000) ranging from 1.58 to 4.87 were used to estimate the gap width from these equations.

[1]Constant should be $-9.61538\text{x}10^{-6}$ instead of $-9.61538\text{x}10^{-5}$ based on code in appendix on page 190 of Halm

2.2.3 Mechanical Losses And Friction

Mechanical losses in scroll compressors contribute directly to a decrease in compressor efficiency, so in order to optimize the scroll compressor, it is necessary to optimize the mechanical losses.

A number of researchers have proposed dynamic models for the scroll compressor, including Morishita (1984), Ishii (1986; 1990; 1996b), Chen (2000) and Liu (2010). The models are all for compressors that use Oldham rings to enforce orientation of the scroll wraps. The dynamic models are all based on applying a force and moment balance to the orbiting scroll and determining the mechanical losses as the sum of the mechanical losses generated by each point of contact.

Few values for the mechanical efficiency are available in open literature, but Ishii has suggested mechanical efficiencies of up to 80% (Ishii et al., 1986) and 92.5% (Ishii et al., 1996b) with friction coefficients of 0.027 and 0.013 respectively.

2.2.4 Heat Transfer

The researchers Ooi and Zhu have conducted computational fluid dynamics (CFD) studies of the heat transfer that occurs in the compression chamber of the scroll compressor. They identified a highly turbulent flow which results in high heat transfer coefficients (Ooi and Zhu, 2004; Zhu and Ooi, 1997) of approximately 4 kW m^{-2} K^{-1}.

Jang and Jeong (2006) have investigated the heat transfer in the scroll compressor and they have found that the temperature of the metal of the scroll involute is very linear with the involute angle. In addition, they proposed a Nusselt number correction for the transverse oscillation of the scroll pocket's wall. Chen (2000) found a similar temperature profile in the scroll wrap.

Dinh and Lear (2005) investigated the impact of adding heat pipes to the fixed scroll of an air compressor in order to cool the air during the working process. They

found that adding heat pipes significantly decreased the component temperatures and decreased the power required for the compressor.

Sunder (1996) investigated the kissing heat transfer between the orbiting and fixed scrolls. This heat transfer is due to the contact of points along the scroll wraps with different temperatures. Sunder (1997) also investigated a simple lumped parameter model which can be used to model scroll compressors but misses much of the complexities of the working process.

Wagner (1995) developed a heat transfer model for the heat transfer in the scroll wraps which includes the wraps and the top and bottom plates. They used the scroll-gas heat transfer coefficient from Myong (1991) for turbulent flow in a rectangular duct. Limited model validation was carried out.

CHAPTER 3. FLOODED VAPOR COMPRESSION CYCLE MODELING

3.1 Motivation

In vapor compression systems, there is an increasing demand for efficient and environmentally-friendly cooling systems. One technology that can offer significant improvements in system efficiency is liquid-flooded compression with regeneration. The analysis presented in this chapter shows that the potential for efficiency increases with flooded compression is significant, and provides motivation for the detailed compressor modeling which follows.

3.2 Baseline Cycle

The standard vapor compression cycle is composed of a compressor to compress the refrigerant up to high pressure, a heat rejection heat exchanger (either gas cooler for supercritical operation or condenser for subcritical operation), a throttling valve to bring the refrigerant back to the evaporation pressure, and an evaporator which allows the refrigerant to evaporate, providing the cooling effect. This configuration is seen in Figure 3.1.

This baseline system has a number of shortcomings, and the main focus of this thesis is to improve the efficiency of this cooling system. The distribution of losses in the system will depend on the exact system configuration, but some of the important losses in the standard vapor compression system are irreversible heat transfer in the heat exchangers, throttling losses in the expansion valve, and irreversibilities in the compression process. Thermodynamic property plots for the baseline cycle are seen in Figure 3.2 for a subcritical carbon dioxide (R744) system.

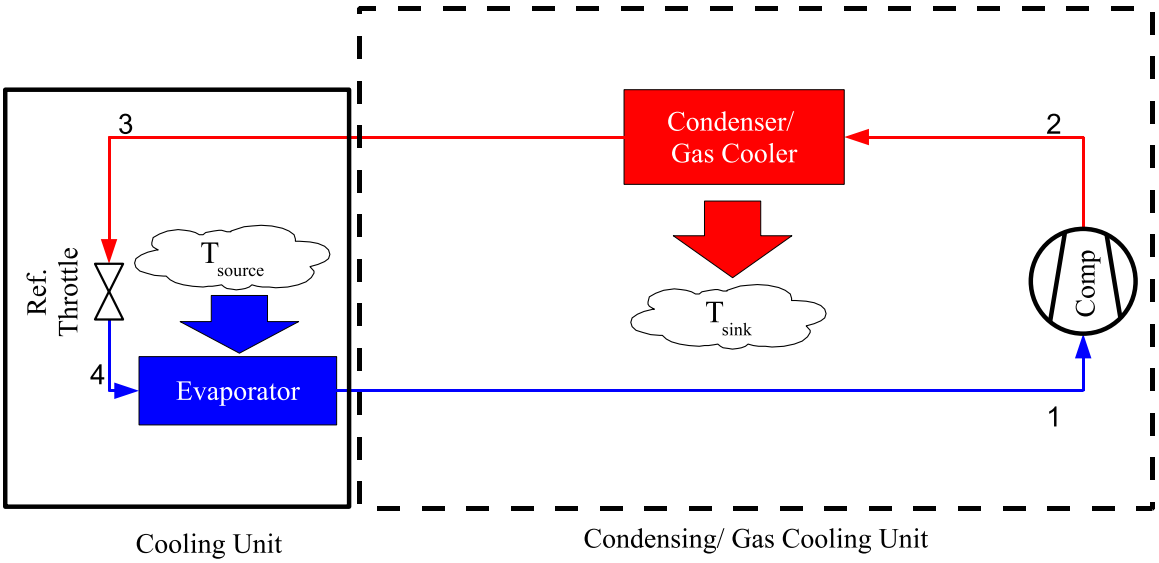


Figure 3.1. Basic Cycle Schematic.

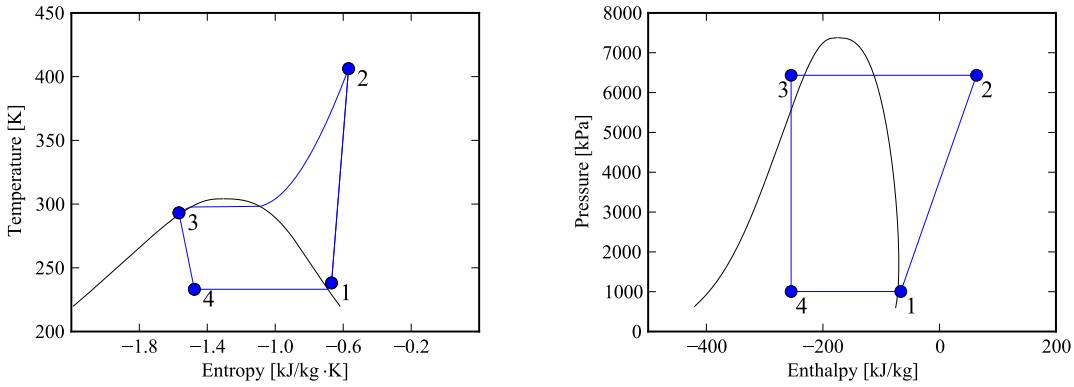


Figure 3.2. Standard Vapor Compression Cycle Property Plots.

One method that has been proposed to improve the performance of the vapor compression cycle is to flood the compressor with a flooding agent, which could be oil or other non-volatile liquids. The motivation for this addition to the standard vapor compression system comes from previous research by Hugenroth (2007) on the Liquid-Flooded Ericsson cycle, which is a gas refrigeration cycle that can approach isothermal compression by flooding the compressor with a large amount of liquid.

This liquid can absorb the heat of compression of the gas which results in an quasi-isothermal compression process. In reality it is not possible to flood the compressor with an infinite amount of oil due to parasitic losses, but the isothermal compression process can be approached.

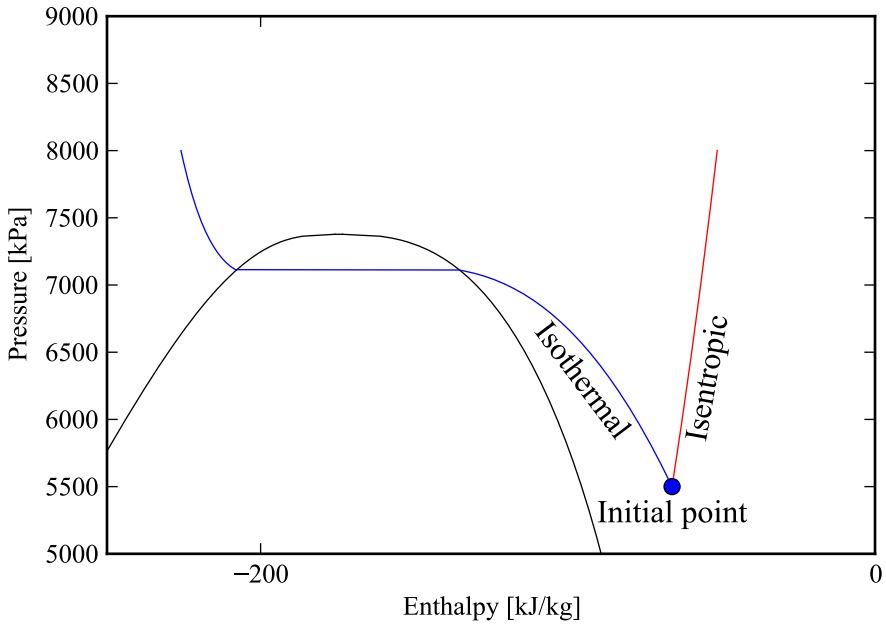


Figure 3.3. Limiting flooded compression processes.

Fundamentally, the application of oil flooding to vapor compression systems is similar to that of gas-cycle refrigeration systems, but the two-phase dome changes the limits on how much oil flooding can be applied. As seen in Figure 3.3, if the refrigerant follows an isothermal compression path from the superheated outlet of the evaporator to the high-side pressure, the refrigerant pathline crosses back into the two-phase region. From a compressor reliability standpoint, as well as an oil separation standpoint, exiting the compressor in the two-phase or subcooled refrigerant regions is not permitted. If the inlet temperature of the compressor is above the critical temperature, it is possible to follow the isotherm into the supercritical region, though at high pressure and low temperature the supercritical fluid behaves more like a

liquid than a gas, making gravitational phase separation difficult. Another theoretical limiting process is that the compression proceeds isentropically, following along a line of constant entropy from low to high pressure.

The addition of oil flooding to a vapor compression system adds some complexity to the system design, as a few more components are needed. As seen in Figure 3.4, the addition of oil flooding adds three major components - an oil separator, an oil cooler and a regenerator. The regenerator is not strictly necessary, but to obtain any benefits from oil-flooding the regenerator is required, as will be further discussed later. Potentially the oil cooler could consist of a few circuits in the gas cooler/condenser, which would minimize extra cost. The oil separator would have to be appropriately designed for the high flow rate of oil and potentially high pressure, but these are surmountable design challenges. In addition, a hydraulic expander can theoretically be used to recover the work of compression of the oil, but the amount of power possible to be recovered in the hydraulic expander is small.

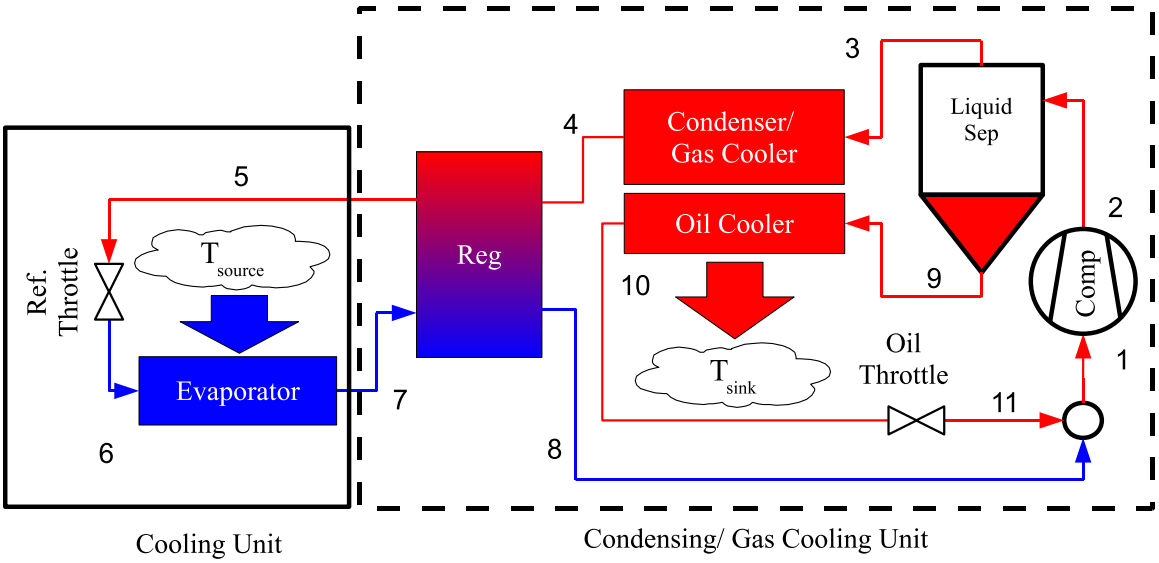


Figure 3.4. Schematic of Flooded Vapor Compression system.

The technology of flooded vapor compression is based on a standard vapor-compression cooling cycle. In this configuration, oil and refrigerant vapor are mixed adiabatically

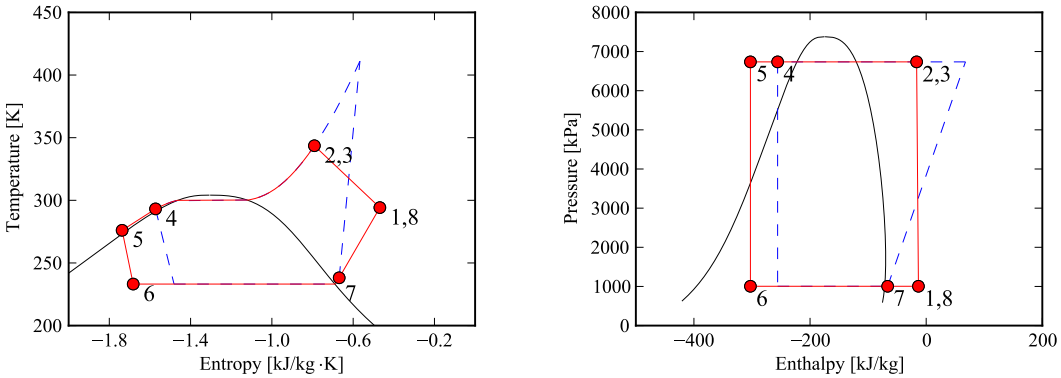


Figure 3.5. Oil Flooded and baseline vapor compression systems (baseline: dashed lines; gas properties shown).

prior to injection into the compressor, and then are compressed together in the compressor from state point 1 to state point 2. After exiting the compressor, the two-phase mixture of liquid and refrigerant vapor enters the high-pressure separator at state point 2, and fully-separated oil and refrigerant vapor streams exit the separator at state points 9 and 3, respectively. The oil and refrigerant are cooled in parallel circuits in the condenser/oil cooler to slightly above the heat sink temperature. The oil exiting the oil cooler at state point 10 is expanded back to state point 11, at which point it is mixed back into the refrigerant vapor exiting the regenerator at state point 8. After being condensed in the condenser, the liquid refrigerant at state point 4 passes through a regenerator where it exchanges heat with the refrigerant vapor exiting the evaporator and the saturated liquid is sub-cooled. The sub-cooled refrigerant at state point 5 is throttled through the expansion device to state point 6 and then evaporated to exit the evaporator at state point 7.

Thermodynamic property plots for the oil flooded with regeneration system are shown in Figure 3.5. The cycle for the baseline system is overlaid in order to demonstrate the benefits of flooded compression. One of the most striking differences with flooded compression is the slope of the compression process. While in the baseline system the gas enthalpy increases in the compression process, in the flooded case the

enthalpy of the gas decreases. While the refrigerant vapor enthalpy decreases during the compression process, the total enthalpy of oil and gas increases in the flooded compression process. The entropy of the refrigerant vapor also decreases during the compression process, but the overall entropy of the mixture increases.

A number of different fluids can be used to flood the system. The constraints on flooding agent selection are that the fluid is liquid throughout the compression process over a wide range of compressor operation points. From thermodynamic arguments Hugenroth (2006) found the best flooding agents were those that had high specific heats as well as high densities. The high density (or low specific volume) is beneficial because it minimizes the flow work on the flooding liquid during the compression process, which can be seen from considering the reversible flow work of an incompressible liquid:

$$|\dot{W}_{rev}| = \dot{m}_l v_l \Delta p \tag{3.1}$$

which shows that the lower the specific volume of the liquid, the lower the reversible compression power for the same flow rate and pressure rise. With regards to the specific heat of the flooding agent, the higher the specific heat of the flooding agent, the less flooding agent is required to absorb the same amount of heat from the refrigerant during the compression process, minimizing the mass flow rate of flooding agent required, and therefore the compression power of the flooding agent. Water therefore is the ideal flooding agent, but likely would struggle to provide sufficient lubricity in the compressor. In addition, water freezes below 0°C. A final factor in the flooding agent selection is the solubility of refrigerant in the flooding liquid. High refrigerant solubility results in parasitic losses as will be described later. In addition, a very viscous flooding liquid will contribute to the pressure drops in the system. With proper design of the liquid loop, the pressure drops should be small as the liquid loop components can be close-coupled with the other components in the condensing unit. Optimization of flooding agent selection remains an open question.

3.3 Cycle Modeling

In order to quantify the benefits of oil flooding it is critical to develop a cycle model which can capture the effects of oil flooding. To simplify the analysis of the cycle and understand the impact of cycle parameters on the performance of the flooded cycle, a number of simplifying assumptions are made:

1. The refrigerant and oil are in mechanical and thermal equilibrium. As will be shown from experimental data in Chapter 6, the assumptions of thermal and mechanical equilibrium are valid due to the highly turbulent mixing of phases in the compression process.

2. Solubility of refrigerant in oil is only considered in the oil separator. In reality, there will always be some amount of refrigerant dissolved in the oil, and this amount will change based on what point of the cycle is being considered. In practice, the only location of the cycle where the solubility is a major concern is in the oil separator. During the compression process, the solubility is not considered since the enthalpy of mixing is small compared to the change of enthalpy in the compression process. In the separator, the solubility of refrigerant in oil is of major importance because the solubility determines the balance of refrigerant going to the evaporator and providing cooling effect versus being solved in the oil and proving no cooling effect. Determining the outlet states of vapor and oil from the separator is challenging as there may be phase inversion, liquid-liquid-vapor equilibrium, or other more complex phase equilibrium problems to contend with. In order to eliminate the challenges of the phase equilibrium problem, it is assumed that some fraction of the mass flow exiting the oil separator is dissolved refrigerant. If the solubility fraction is zero, all the refrigerant exits the oil separator in the vapor phase. Where solubility data is available, it is used to predict the equilibrium solution oil mass fractions.

3. Fixed superheat, subcooling, and the pinch temperatures between the outlet of the heat exchangers and their respective thermal reservoir. Fixing the superheat

captures the performance of a system operating with an ideal thermostatic expansion valve (TXV). Fixing the subcooling means that the charge is floating, but allows for an understanding of flooding for different charge levels indirectly by considering the performance for different fixed subcooling amounts.

4. Fixed compressor volumetric displacement with constant volumetric efficiency of 100%.

3.3.1 Mixture Properties

In the sections which follow, simple models are presented for each of the cycle components. All the components that are flooded require mixture properties of the refrigerant-oil mixture. For the simplified analysis presented here, only two mixture thermodynamic properties are required - the enthalpy of the mixture and the entropy of the mixture. If more detailed models were employed for the components, transport mixture properties would also be required. For an ideal mixture of two phases, the mixture enthalpy and mixture entropy can be given by

$$h_m = x_l h_l + x_g h_g \tag{3.2}$$

$$s_m = x_l s_l + x_g s_g \tag{3.3}$$

which is simply an oil-mass-fraction weighted average of the properties of the individual phases at the overall temperature and pressure. The oil and gas mass fractions are given by

$$x_l = \frac{\dot{m}_l}{\dot{m}_l + \dot{m}_g} \tag{3.4}$$

and $x_g = 1 - x_l$ respectively.

The total mass flow rate passing through the compressor is driven by the specific volume of the mixture in the compressor displacement volume. From two-phase flow fluid dynamics, the local cross-sectional area is given by

$$\alpha = \frac{1}{1 + \frac{x_l}{x_g}\frac{\rho_g}{\rho_l}K} \tag{3.5}$$

where K is the slip ratio of the phases. The mixture specific volume can therefore be given by

$$v_m = \frac{x_g v_g + K x_l v_l}{x_g + K x_l} \tag{3.6}$$

where the slip ratio K can be given by the Zivi (1964) slip model, or

$$K = \left(\frac{v_g}{v_l}\right)^{1/3}. \tag{3.7}$$

For positive displacement compressors like scroll compressors, the gas can be assumed to be sufficiently slowed in the compressor working pocket such that the speeds of the vapor and liquid phases can be assumed to be equal (so called homogeneous flow), yielding a slip ratio of

$$K = 1. \tag{3.8}$$

Homogeneous flow was assumed for all the analysis which follows, which yields the mixture specific volume of

$$v_m = x_g v_g + x_l v_l \quad (homogeneous). \tag{3.9}$$

The properties of the refrigerants are obtained from the equations of state listed in Appendix C.2. The oil properties are significantly more difficult to obtain as the manufacturer data is quite sparse, and sometimes does not include the specific heat, a critical parameter in the thermodynamic analysis. Appendix C.4 lists the manufacturer data employed for the oil properties. When oil properties are not available, as a rough approximation the specific heat can be obtained from Liley and Gambill (1973)

$$c_{p,l} = 4.186 \frac{0.388 + 0.00045(1.8T + 32)}{SG^{0.5}} \tag{3.10}$$

where T is in °C, $c_{p,l}$ is in kJ/kg-K, and SG is the specific gravity of the oil at 15.56 °C. This equation is valid for petroleum oils in the range of temperature between -10 °C and 204 °C and the range of specific gravity 0.75 to 0.96. Thome (2004) recommends Eqn. (3.10) be used for higher specific gravity oils in the absence of other tabulated data. A set of thermophysical data for a range of oils, liquids and other fluids can be found in the VDI Wärmeatlas (Hunold, 2010) and other fluids can be found in Melinder's work (2010)

3.3.2 Compressor

The compressor is modeled as having a constant adiabatic efficiency based on the mixture properties. As will be shown in Chapter 6, the adiabatic efficiency for compressors not appropriately designed for oil flooding tends to decrease with an increase in oil mass fraction. With proper redesign, as shown in Chapter 7, the decrease in adiabatic efficiency with more oil flooding can be greatly decreased. The inlet state point for the compressor can be determined from mixture properties by

$$h_1 = h_m\left(T_1, p_1, x_{l,1}\right) \tag{3.11}$$

$$s_1 = s_m\left(T_1, p_1, x_{l,1}\right) \tag{3.12}$$

while the isentropic outlet enthalpy can be determined from

$$h_{2s} = h_m\left(p_2, s_1, x_{l,1}\right) \tag{3.13}$$

which must be iteratively determined by a numerical solver. The adiabatic efficiency is defined from

$$\eta_{comp} = \frac{h_2|_{s=s_1} - h_1}{h_2 - h_1}. \tag{3.14}$$

This definition of adiabatic efficiency assumes that there is no external heat transfer which is a good assumption for compressors flooded with large amounts of oil since the oil will tend to decrease the discharge temperature of the compressor and therefore the amount of heat lost to the ambient through convection and radiation. This then yields the discharge entropy of

$$s_2 = s_m\left(p_2, h_2, x_{l,2}\right) \tag{3.15}$$

where the inlet $x_{l,1}$ and outlet $x_{l,2}$ oil mass fractions are the same. All the mass flow paths pass through the compressor, so the compressor sees the highest mass flow rate of any component. Thus the compressor power input can be given by

$$\dot{W}_{comp} = \left(\dot{m}_l + \dot{m}_g\right)\left(h_2 - h_1\right) \tag{3.16}$$

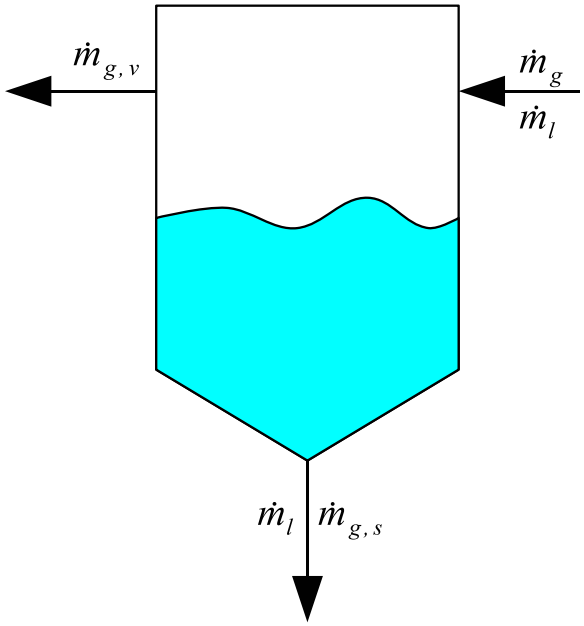


Figure 3.6. Schematic of mass flows in oil separator.

and the irreversibilities generated in the compression process are given by

$$\dot{E}_{comp} = (\dot{m}_l + \dot{m}_g)T_0(s_2 - s_1) \tag{3.17}$$

where T_0 is the reference temperature, taken to be 298 K.

3.3.3 Oil Separator

In an ideal separation process, a high-pressure mixture of oil and gas enters and is split into pure vapor and oil streams by gravitational separation. In practice there will always be some refrigerant dissolved in the oil, and at least some quantity of oil in the refrigerant vapor in droplet form. Through proper sizing and design of the oil separator, the amount of oil carry-over in the refrigerant vapor can be reduced to a negligible amount. Solubility of refrigerant in oil is significantly more difficult to eliminate as it is fundamentally governed by the inter-molecular interactions of refrigerant and oil, and driven by the pairing of flooding liquid with refrigerant. Of the refrigerant that enters the separator ($\dot{m}_g$), some amount ($\dot{m}_{g,s}$) exits the separator

solved in the refrigerant, and the remainder exits in the gas phase ($\dot{m}_{g,v}$). Figure 3.6 shows a schematic of the flows in the oil separator. Solubility will be further discussed in Section 3.4.2. Therefore the mass fraction of the liquid stream at the outlet of the separator that is solved refrigerant is defined by

$$x_{g,s} \equiv \frac{\dot{m}_{g,s}}{\dot{m}_{g,s} + \dot{m}_l} \tag{3.18}$$

and thus the mass flow rate of refrigerant solved in the oil is

$$\dot{m}_{g,s} = \dot{m}_l \frac{x_{g,s}}{1 - x_{g,s}} \tag{3.19}$$

and the oil mass fraction of the oil in the oil-refrigerant mixture exiting the separator is

$$x_{l,s} = 1 - x_{g,s}. \tag{3.20}$$

The separation is assumed to take place isothermally, thus

$$T_3 = T_9 = T_2 \tag{3.21}$$

$$p_3 = p_9 = p_2. \tag{3.22}$$

Some solubility data (for instance the solubility of CO_2 in water) is given in terms of molar fractions, so from this it is possible to convert back to mass fractions by applying

$$x_{g,s} = \frac{1}{1 + \left(\frac{1 - y_g}{y_g}\right) \frac{M_l}{M_g}} \tag{3.23}$$

where y_g is the molar fraction of gas and M_l and M_g are the mole masses of liquid and gas respectively.

3.3.4 Condenser/Gas Cooler

Depending on whether the condensing pressure is above the critical pressure or not, the cycle heat rejection can either take place as a condensation process or supercritical heat rejection. In the case of a condenser, the pinch point in the heat

rejection is defined to be between the condenser outlet temperature and the ambient air temperature. If a supercritical gas cooler is present, the pinch temperature is defined to be the difference in temperature between the gas cooler outlet temperature and the ambient temperature. Thus in either case, the temperature at point 4 (outlet of hot heat exchanger) is given by

$$T_4 = T_{sink} + \Delta T_{pinch} \tag{3.24}$$

and the enthalpy and entropy are given by

$$\begin{aligned} h_4 &= h(p_4, T_4, x_l = 0) \\ s_4 &= s(p_4, T_4, x_l = 0). \end{aligned} \tag{3.25}$$

In the case of subcritical heat rejection (condensation), p_4 is uniquely defined based on the saturation temperature at the condensing temperature, while for supercritical heat rejection, the gas cooler pressure is unknown and is determined through optimization of the coefficient of performance of the cycle. For subcritical heat rejection, the condensing temperature is given by

$$T_{cond} = T_{sink} + \Delta T_{pinch} + \Delta T_{sc} \tag{3.26}$$

where ΔT_{sc} is the quantity of subcooling. The amount of heat rejected is equal to

$$\dot{Q}_{reject} = \dot{m}_{g,v} \left(h_4 - h_3 \right) \tag{3.27}$$

and the irreversibilities generated in the heat rejection process are equal to

$$\dot{E}_{reject} = T_0 \left[\dot{m}_{g,v} (s_4 - s_3) - \frac{\dot{Q}_{reject}}{T_{sink}} \right]. \tag{3.28}$$

3.3.5 Oil Cooler

The oil cooler cools the oil and some solved refrigerant against the heat sink medium, and it is considered that the oil is cooled to within ΔT_{pinch} of the ambient temperature, so

$$T_{10} = T_{sink} + \Delta T_{pinch}. \tag{3.29}$$

The oil loop may have some solved refrigerant in it which complicates the analysis for the "oil" cooling process. The oil cooler process is broken up into two parts; the refrigerant and the liquid are treated as going through parallel processes. In this way, the heat rejected by the liquid is given by

$$\dot{Q}_{oil-cooler,l} = \dot{m}_l c_{p,l} \left(T_9 - T_{10}\right) \tag{3.30}$$

and the irreversibilities in the liquid heat rejection process by

$$\dot{E}_{oil-cooler,l} = T_0 \left(\dot{m}_l (s_{9,l} - s_{10,l}) - \frac{\dot{Q}_{oil-cooler,l}}{T_{sink}} \right). \tag{3.31}$$

The solved refrigerant goes through a heat rejection process which is assumed to occur at constant pressure. Like the primary refrigerant stream, it may experience a phase change during the heat rejection if the temperature is below the critical temperature. The heat of mixing can be neglected (Conde, 1996), and thus the analysis is exactly the same as for the condenser/gas cooler except that the mass flow rate is $\dot{m}_{g,s}$ rather than $\dot{m}_{g,v}$. Thus the amount of heat rejected is

$$\dot{Q}_{oil-cooler,g} = \dot{m}_{g,s} \left(h_{10,g} - h_{9,g}\right) \tag{3.32}$$

and the irreversibilities generated in the heat rejection process are equal to

$$\dot{E}_{oil-cooler,g} = T_0 \left[\dot{m}_{g,s} (s_{10,g} - s_{9,g}) - \frac{\dot{Q}_{oil-cooler,g}}{T_{sink}} \right]. \tag{3.33}$$

3.3.6 Hydraulic Expansion Device

In the hydraulic expansion device, liquid and a small amount of solved refrigerant are expanded together to the low pressure side of the system. If a hydraulic expander is used, some of the compression power of the liquid can be recovered. As in the oil cooler, the refrigerant and liquid streams are assumed to be go through parallel

working processes. For the oil, with an assumption of incompressible oil, the amount of recovered power is equal to

$$\dot{W}_{hyd-exp,l} = \eta_{hyd-exp}\dot{m}_l v_{10}\left(p_{10} - p_{11}\right) \tag{3.34}$$

where the limiting expansion process is an adiabatic reversible expansion of incompressible oil. Thus the outlet conditions can be determined from

$$\dot{W}_{hyd-exp,l} = \dot{m}_l\left(h_{10,l} - h_{11,l}\right) \tag{3.35}$$

where the enthalpies $h_{10,l}$ and $h_{11,l}$ are based on pure liquid properties.

For the refrigerant, the ideal working process is an isentropic expansion process, but in practice the working process will not be isentropic. Thus the outlet enthalpy of the expansion process can be obtained from

$$\eta_{hyd-exp,g} = \frac{h_{11,g} - h_{10,g}}{h_{11,g}|_{s=s_{10,g}} - h_{10,g}} \tag{3.36}$$

where $h_{11,g}|_{s=s_{10,g}}$ is the isentropic enthalpy. The work recovered by the refrigerant is given by

$$\dot{W}_{hyd-exp,g} = \dot{m}_{g,s}(h_{11,g} - h_{10,g}) \tag{3.37}$$

and the irreversibilities generated by

$$\dot{E}_{hyd-exp,g} = T_0\dot{m}_{g,s}(s_{11,g} - s_{10,g}). \tag{3.38}$$

If the hydraulic expander is not used in the system configuration, the adiabatic efficiency of the hydraulic expander is set to 0, and thus the hydraulic expander becomes a throttling valve.

3.3.7 Evaporator And Expansion Valve

In the expansion valve, the refrigerant is throttled down to the evaporating pressure at constant enthalpy. In order to determine the saturation temperature, there is again assumed to be a pinch temperature of ΔT_{pinch} between the exit of the evaporator and the ambient. Thus if the outlet gas exiting the evaporator is superheated, the evaporation temperature can be obtained from

$$T_{evap} = T_{source} - \Delta T_{pinch} - \Delta T_{sh} \tag{3.39}$$

and the enthalpy at the outlet to the evaporator can be obtained from

$$h_7 = h_m \left(T_{evap} + \Delta T_{sh}, p_{evap}, x_l = 0 \right). \tag{3.40}$$

The evaporator saturated suction pressure can be determined from the evaporation temperature. Thus the evaporator capacity is given by

$$\dot{Q}_{evap} = \dot{m}_{g,v} \left(h_7 - h_6 \right) \tag{3.41}$$

and the irreversibilities generated are equal to

$$\dot{E}_{evap} = T_0 \left[\dot{m}_{g,v}(s_7 - s_6) - \frac{\dot{Q}_{evap}}{T_{source}} \right] \tag{3.42}$$

where the heat transfer is assumed to all occur at a temperature equal to the source temperature. The inlet entropy can be determined from knowledge of the evaporation temperature and the inlet enthalpy.

3.3.8 Regenerator

In the regenerator, the gas exiting the evaporator is used to cool the refrigerant exiting the condenser/gas cooler. The flow streams in the regenerator have the same mass flow rate so the maximum amount of heat transfer per unit mass flow in the regenerator is equal to

$$\Delta h_{Reg,max} = min \begin{cases} h_4 - h(T_7, p_4) \\ h(T_4, p_7) - h_7 \end{cases}. \tag{3.43}$$

The two terms in the right hand side of Eqn. (3.43) are the constant pressure heating or cooling of each stream to the inlet temperature of the other stream. Typically the capacitance rate of the gas stream is less than that of the refrigerant exiting the condenser/gas cooler because the specific heat of vapor is less than that of liquid,

and thus the evaporator outlet stream is also the limiting enthalpy change. The effectiveness of the regenerator is given by

$$\varepsilon_{Reg} = \frac{\Delta h_{Reg}}{\Delta h_{Reg,max}}. \tag{3.44}$$

Thus the enthalpies of each stream exiting the regenerator are equal to

$$\begin{aligned} h_5 &= h_4 - \Delta h_{Reg} \\ h_8 &= h_7 + \Delta h_{Reg} \end{aligned} \tag{3.45}$$

and the irreversibilities in the regeneration process are equal to

$$\dot{E}_{Reg} = T_0 \dot{m}_{g,v}[(s_8 + s_5) - (s_4 + s_7)]. \tag{3.46}$$

3.3.9 Mixer

The oil and solved refrigerant exiting the hydraulic expander then mixes with the gas exiting the regenerator. Since the capacitance rate of the oil stream is higher than that of the gas stream, required so that isothermal compression can be achieved, the gas-oil mixture takes on nearly the outlet temperature of the oil. The mixing process is modeled as being adiabatic, and thus the outlet enthalpy h_1 can be obtained from

$$(\dot{m}_l + \dot{m}_{g,s})h_{11} + \dot{m}_{g,v}h_8 - (\dot{m}_l + \dot{m}_{g,v} + \dot{m}_{g,s})h_1 = 0 \tag{3.47}$$

where the irreversibilities of the mixing process are found from

$$\dot{E}_{mixer} = T_0 \begin{bmatrix} (\dot{m}_l + \dot{m}_{g,v} + \dot{m}_{g,s})s_1 - \dot{m}_{g,v}s_8 \\ -(\dot{m}_l + \dot{m}_{g,s})s_{11} \end{bmatrix}. \tag{3.48}$$

3.3.10 Closure

In order to solve the system of equations presented above, the equations were implemented into the Python programming language, and the plotting package matplotlib (Hunter, 2007) was used to generate the figures. The code can be found in

Appendix A. A secant loop is used to solve for the inlet temperature to the compressor, in which the compressor inlet temperature is modified in order to enforce the energy balance on the overall system. In addition, the total mass flow rate of the compressor is determined based on the displacement rate of the compressor and the mixture density at the compressor inlet. The flooded vapor compression system can be run in either heat pump or cooling mode - for both configurations the cycle analysis remains exactly the same but the figures of merit are different. The system COP in cooling mode is

$$COP_{AC} = \frac{\dot{Q}_{evap}}{\dot{W}_{comp} + \dot{W}_{hyd-exp}} \tag{3.49}$$

and in heating mode, the COP is

$$COP_{HP} = \frac{-\dot{Q}_{cond}}{\dot{W}_{comp} + \dot{W}_{hyd-exp}}. \tag{3.50}$$

The second law efficiencies of the systems can also be defined from

$$\eta_{II,AC} = \frac{COP_{AC}}{\left(\frac{T_{source}}{T_{sink} - T_{source}}\right)} \tag{3.51}$$

$$\eta_{II,HP} = \frac{COP_{HP}}{\left(\frac{T_{sink}}{T_{sink} - T_{source}}\right)}. \tag{3.52}$$

3.4 Modeling Results

In this section, modeling results are presented for a range of working fluids, flooding agents, solubility levels, and component efficiencies. To begin, the working fluid is varied and all other parameters are fixed in order to determine the most promising fluids for use with liquid flooding. After determining good candidate working fluids, a range of flooding agents are modeled in order to find the flooding agents that offer the best system performance. Once good working-fluid/flooding-agent pairs have been determined, the impact of solubility in the phase separator is included. Finally after the model development is complete, the model is used to predict the performance of a number of heat-pump cycles operating in different conditions.

3.4.1 Fluid Pair Selection

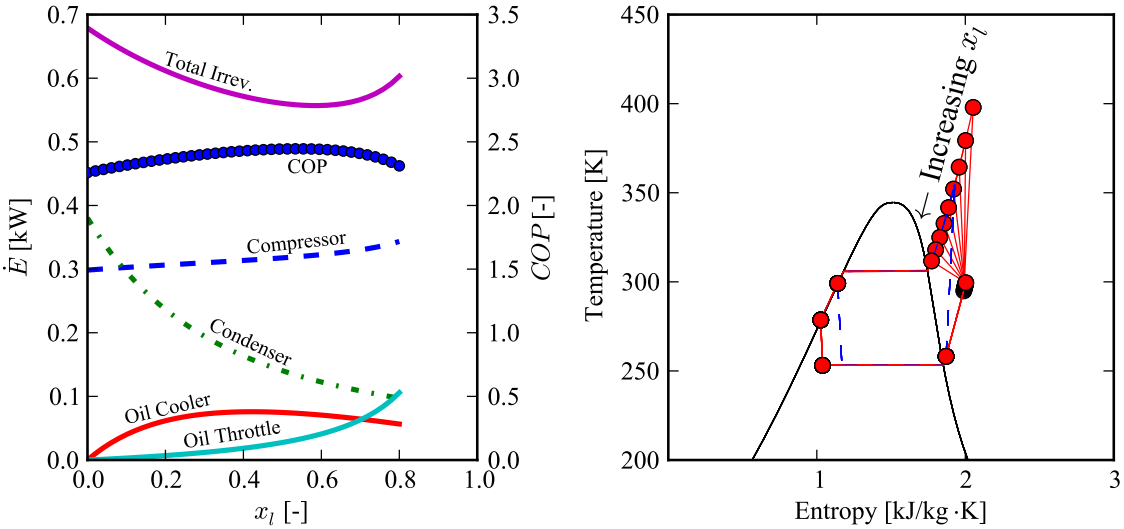
For fixed heat source (cool space) and heat sink (ambient) temperatures as well as superheat, subcooling and pinch temperature differences, the suction pressure, suction temperature, and discharge pressure of the compressor are fixed. For initial fluid-pair evaluations, the separation process is assumed to be complete with no refrigerant solubility in the flooding liquid. The parameters of Table 3.1 were used unless otherwise specified. The pinch, subcooling, and superheat values were obtained from the default input parameters from the Oak Ridge National Labs Heat Pump Simulation Model (Rice, 2005) and are believed to be representative values.

Table 3.1 Default parameters for flooded cycle modeling.

Parameter	Value
$\dot{V}_{comp}$ [m^3/h]	4.32
ΔT_{pinch} [K]	5.0
$\Delta T_{subcool}$ [K]	7.0
$\Delta T_{superheat}$ [K]	5.0
η_{comp} [-]	0.7
$\eta_{hyd-exp}$ [-]	0.0
ε_{Reg} [-]	0.9
$x_{g,s}$ [-]	0.0

Therefore, the only free parameter affecting the compression power is the liquid mass flow fraction. The liquid flow rate impacts five irreversibility terms - the irreversibilities in the liquid expansion device, the irreversibilities in the compressor, the irreversibilities in the condenser (gas cooler in the case of supercritical CO_2), the irreversibilities in the liquid cooler, and the irreversibilities in the mixer. For a R410A cooling cycle operating between source and sink temperatures of -10°C and 26.7°C respectively, the maximum system COP will be obtained for the oil mass flow fraction

which minimizes the system irreversibilities. PAG oil was employed as the flooding agent. Figure 3.7(a) demonstrates that as the oil mass flow fraction increases, both the compressor and oil expansion device irreversibilities increase since neither the compression or expansion of the oil proceeds isentropically, and more oil flow results in more irreversibility. Even though the oil helps to decrease the discharge temperature and approach an isothermal compression process, there is still an increase in compression irreversibility. The reason is that the oil still needs to be compressed in the compressor and the irreversibilities in the oil compression process are significant. On the other hand, the condenser irreversibilities are decreased since the desuperheating horn is reduced due to a more isothermal compression process. Typical oil mass flow fractions for non-flooded systems are on the order of 1% by mass. In the oil cooler, the irreversibilities increase initially due to the increase in oil flow rate, but reach a maximum and begin to decrease as the oil cooler inlet temperature decreases as a result of the flooding.



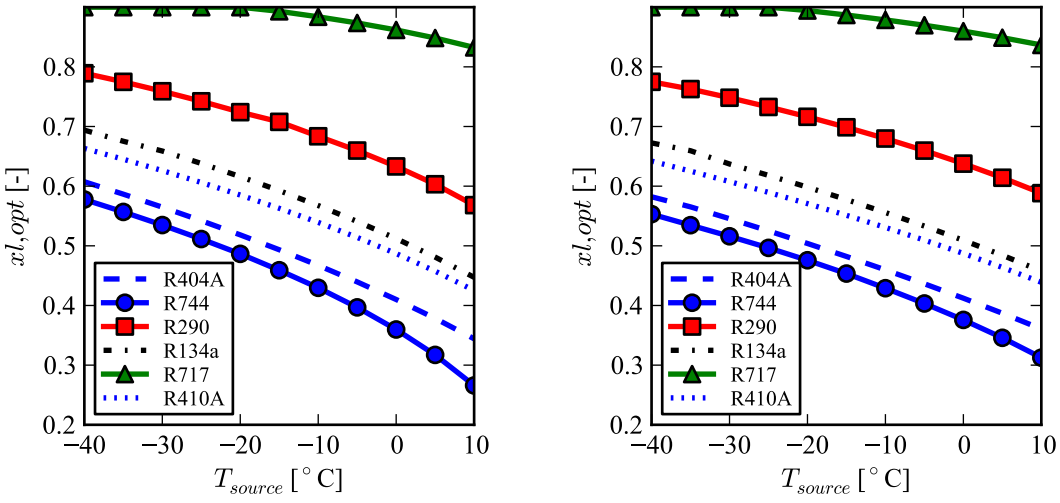
(a) Component irreversibilities and cycle COP

(b) T-s plot as a function of x_l

Figure 3.7. R410A losses and R410A T-s plot with varied oil flooding rates for T_{source} = -10°C, T_{sink} = 26.7°C, and PAG oil.

Figure 3.7(b) shows that the inlet temperature to the compressor is effectively fixed due to the combination of regeneration and oil flooding. Since the minimum capacitance rate for the regenerator is the stream exiting the evaporator, this stream is the one that sees the largest increase in enthalpy, which means that for a 90% effective regenerator, the low-pressure outlet temperature of the regenerator will always be 90% of the way from the evaporator outlet temperature to the condenser outlet temperature. Thus the compressor inlet temperature is strongly linked with the condenser outlet temperature. The rest of the variation in the compressor inlet temperature is linked to the amount of oil. Larger oil mass fractions will tend to drive the compressor inlet temperature closer to the oil temperature, which is also near the heat sink temperature since the oil is cooled against the sink temperature.

Similarly, as the oil flow rate is increased, the compression process becomes more and more isothermal. Oil mass flow fractions from 0 to 0.8 are plotted. Further increasing the oil mass fraction would result in a discharge temperature that was in the two-phase dome. The optimal liquid mass flow fraction is dependent on the working



(a) $T_{sink} = 26.7°C$ (b) $T_{sink} = 45.0°C$

Figure 3.8. Optimal oil mass fraction for systems flooded with PAG oil.

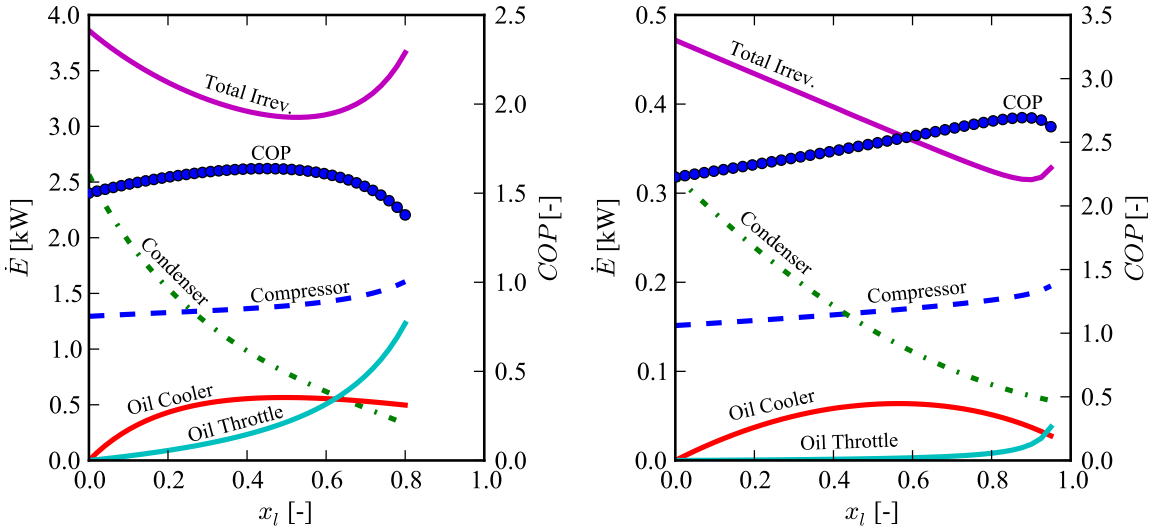
fluid employed. Figure 3.8 shows the optimal oil mass fractions for a variety of refrigerants for two sink temperatures and a range of source temperatures. The optimal oil mass fraction is found using the *fminbound()* routine provided the ScientificPython (SciPy) package. The oil mass fraction was bound by the limits 0 and 0.9, and only ammonia hit the bound values. For CO_2, the COP of the flooded system was optimized by varying the gas cooler pressure and oil mass fraction with the L-BFGS-B algorithm (Byrd et al., 1995) where the gas cooler pressure is bound to be supercritical and the oil mass fraction is bound to fall between 0 and 0.8. Futher discussion of CO_2 with flooding is found below. The refrigerants selected are either in common use or are natural refrigerants. The natural refrigerants are marked with solid lines. The range in optimal oil mass fractions is driven by the particular thermodynamics of the fluid as well as the shape of the two-phase dome.

The wide range in optimal oil mass fractions can be better understood by considering the irreversibilities for two of the fluids with the greatest disparity in optimal oil mass fraction - CO_2 and ammonia. Figure 3.9 shows the component irreversibilities as a function of oil mass fraction for both CO_2 and ammonia. For CO_2, since the heat rejection pressure is supercritical, the optimal oil mass fraction and the gas cooler pressure are free variables, but here the gas cooler pressure was fixed at 100 bar for ease of comparison. The vast difference in optimal oil mass fraction can now be understood as being partly driven by the large difference between suction and discharge pressures for the refrigerant. Table 3.2 shows the properties for all the fluids investigated here.

The large difference in pressure over the compressor for CO_2 means that the oil experiences a large pressure lift in the compressor. As a result, the oil contributes greatly to the compression and expansion irreversibilities due to the large difference in pressure. The penalty for the oil compression/expansion for ammonia is far lower since the difference in pressure for ammonia is five times lower than that of CO_2. Thus the optimal oil mass fraction for ammonia is higher.

Table 3.2 Properties of working fluid in generic compression process (Compressor operating with T_{evap}=5°C, ΔT_{sh}=1°C, T_{cond}=28°C).

Fluid	ρ	c_p	p_{ratio}	Δp
-	kg/m^3	kJ/kg-K	-	kPa
Ammonia	4.1	2.74	2.13	584
Propane	12.0	1.83	1.86	476
R134a	17.1	0.93	2.08	377
R404A	35.5	1.00	1.91	640
R410A	35.6	1.16	1.91	850
CO_2	112.9	2.06	1.74	2922



(a) CO_2 with p_{gc}=100 bar

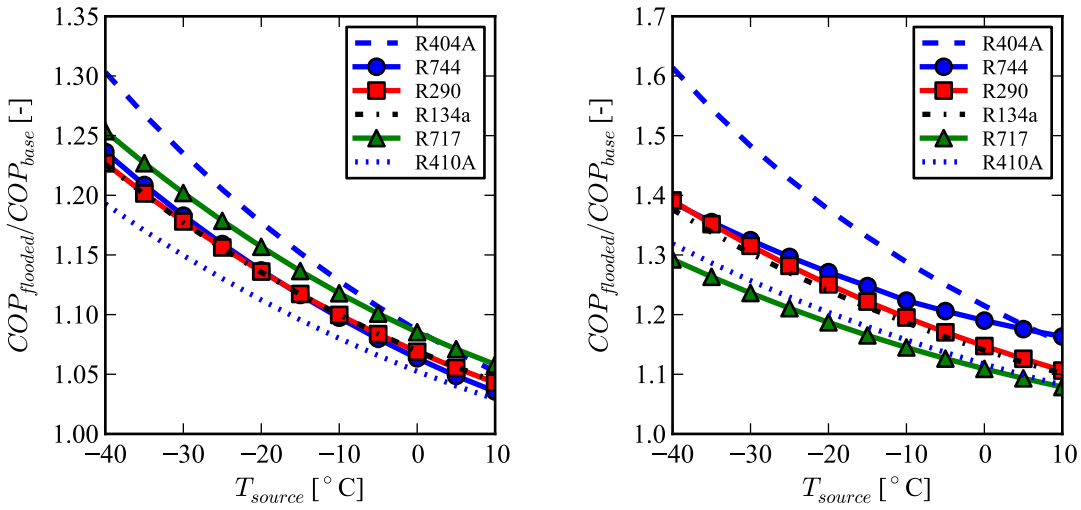
(b) Ammonia

Figure 3.9. CO_2 and ammonia component irreversibilities and system COP as a function of oil mass fraction for T_{source}=-10°C, T_{sink}=26.7°C, and PAG oil with no refrigerant solubility.

The critical parameter for oil flooding is the increase in system Coefficient of Performance in cooling mode with oil flooding. Figure 3.10 shows the ratio of the flooded

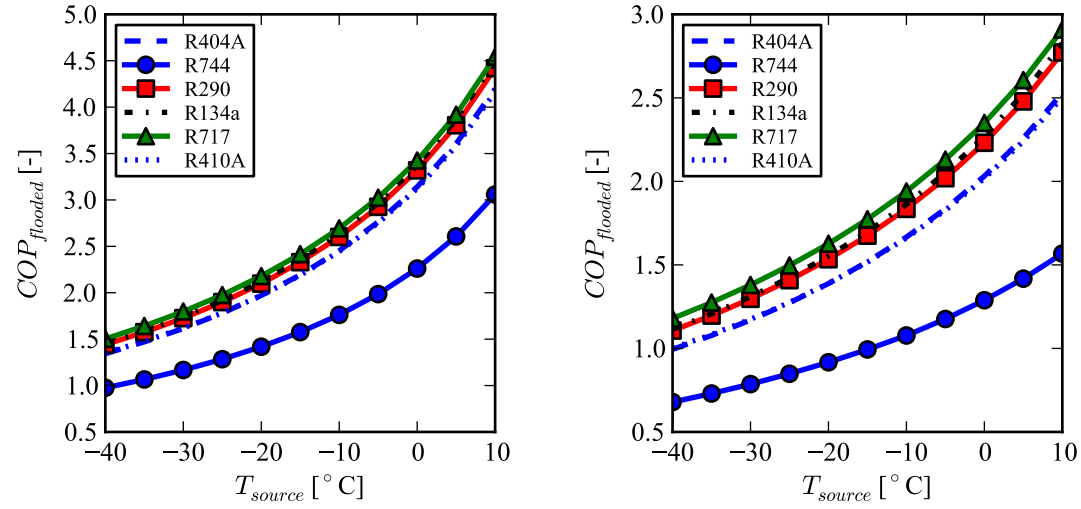
COP to the baseline COP for two different sink temperatures and a range of source temperatures for a range of different working fluids. The oil mass fraction of the flooded system is optimized, and for CO_2, the baseline system's gas cooler pressure is optimized and the flooded system's oil mass fraction and gas cooler pressure are optimized since the heat rejection is always supercritical with these sink temperatures. The fluid that experiences the greatest improvement in cycle efficiency with liquid flooding is R404A. For a source temperature of 26.7°C, all the systems see an increase in COP, and as the source temperature decreases, the increase in COP gets larger. This increase in improvement at lower source temperatures is due to the increase in size of the superheat horn for the baseline system as the source temperature decreases - an irreversibility that liquid flooding can effectively decrease. As the temperature lift of the system increases, the increase in COP also increases. For a sink temperature of 45°C (the highest temperature experienced in Phoenix, Arizona in a typical year is 44°C), the improvement in system COP with liquid flooding also increases above that of cooler ambient conditions. For supermarket applications in high ambient temperature conditions, liquid flooding can be particularly beneficial. In supermarket applications, it is necessary to keep goods frozen at low temperatures, even in high ambient temperature conditions. Refrigerant R404A is commonly used for these applications, which suggests an opportunity for significant improvements in system COP with minimal investment. At a source temperature of -30°C and a sink temperature of 45°C, the increase in COP for refrigerant R404A is nearly 50%.

The absolute COP of the systems with flooding is ultimately what drives the energy consumption of flooded compression systems. Figure 3.11 presents the COP of the systems with liquid flooding. Again oil mass fractions are optimized and the gas cooler pressure is optimized as well for CO_2. From these results it is clear that flooded system efficiency for the majority of the refrigerants are clustered tightly together while CO_2 is significantly lower. The best system COPs are achieved with the refrigerants ammonia, R134a and propane.



(a) $T_{sink} = 26.7°C$ (b) $T_{sink} = 45.0°C$

Figure 3.10. Ratio of flooded system COP to baseline system COP flooded with PAG oil and no refrigerant solubility for a range of refrigerants.



(a) $T_{sink} = 26.7°C$ (b) $T_{sink} = 45.0°C$

Figure 3.11. Flooded COP for PAG oil.

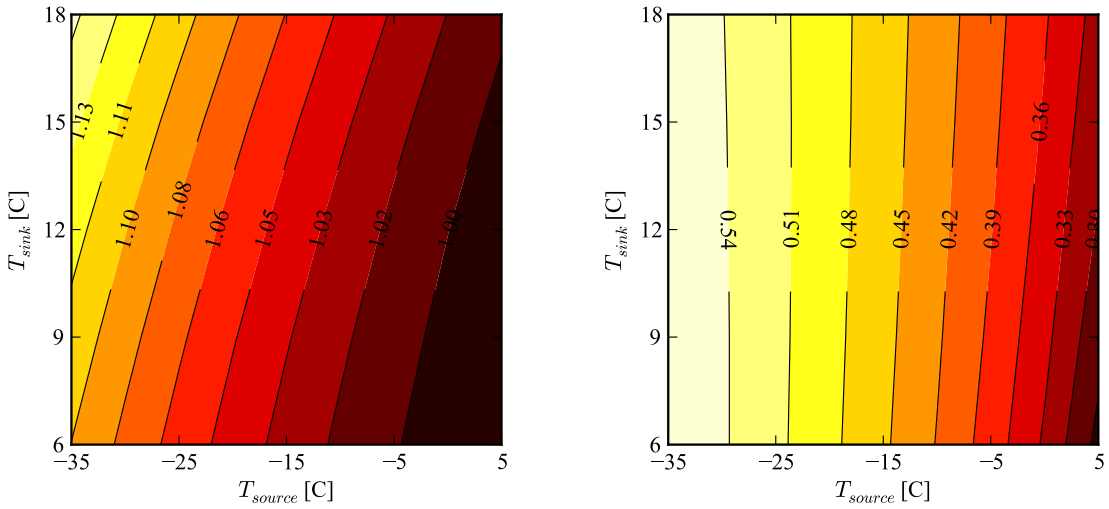
While propane, ammonia, and R134a can achieve the best efficiency with liquid flooding, the analysis that follows will focus on R410A and CO_2. CO_2 is of interest because it is a naturally occurring refrigerant whose environmental impacts are very well understood. R410A is also of interest due to its prevalence in many residential air-conditioning and heat pumping applications. Further work should continue the development of liquid-flooded systems for the refrigerants R134a and propane because the improvement in their system COP is significant with liquid flooding.

3.4.2 CO_2 With Flooding

The goal of this section is to expand on the analysis presented above for the refrigerant CO_2 and include some of the other features described. In particular, it is necessary to add the impact of solubility of CO_2 in the liquid flooding agent and a study on the selection of the flooding agent.

To begin, it is first important to understand the differences between CO_2 and other refrigerants. CO_2 has a critical temperature of 30.98°C, which is quite low compared with, for example, R410A at 71.35°C. This means that if there is 5°C pinch as well as a 7°C subcooling, it is only possible to condense CO_2 at ambient temperatures below 18.98°C. The transition to transcritical operation typically results in a significant penalty to system efficiency. For this reason, one of the most prominent applications for CO_2 as a refrigerant currently is as the low-temperature side of a cascade system, often with ammonia as the high-temperature refrigerant (Pearson, 2005).

The flooded system optimization procedure was carried out over a range of subcritical operating temperatures, and the optimum oil mass fraction was found for each source/sink temperature pair. The results of this subcritical flooding analysis are shown in Figure 3.12. The optimal oil mass fraction is quite sensitive to the source temperature, but is only a weak function of the sink temperature. On the other hand, the cycle COP improvement is greatest for the largest temperature lift; for the highest temperature lift (-35°C/18°C), the improvement in system COP is greater than 14%.



(a) Contour plot of the ratio of COP of flooded systems to baseline systems as a function of the source and sink temperatures.

(b) Contour plot of the optimal oil mass fraction of flooded systems as a function of the source and sink temperatures.

Figure 3.12. Subcritical CO_2 systems flooded with PAG oil.

At high source temperatures, the decrease in desuperheating losses is not offset by an increase in cooling capacity, resulting in a decrease in COP.

Once the ambient temperature is high enough that the refrigerant can no longer condense, the system is called a transcritical cycle. This means that the evaporating pressure is below the critical pressure and the heat rejection occurs at a pressure above the critical pressure. If the outlet temperature of the gas cooler is then fixed, there are a family of cycles with different gas cooler pressures. Figure 3.13 shows a selection of different transcritical CO_2 cycles with varied gas cooler pressures and an evaporator outlet fixed at the saturated state. The gas cooler outlet temperature is fixed at 26.9°C (300 K). As the gas cooler pressure increases, both the refrigerating effect and the compression power increase. As a result there is an optimal gas cooler pressure which optimizes the efficiency of the transcritical CO_2 cycle.

Figure 3.14 shows a contour plot of the transcritical flooded system COP as a function of both oil mass fraction and gas cooler pressure. From these results it is

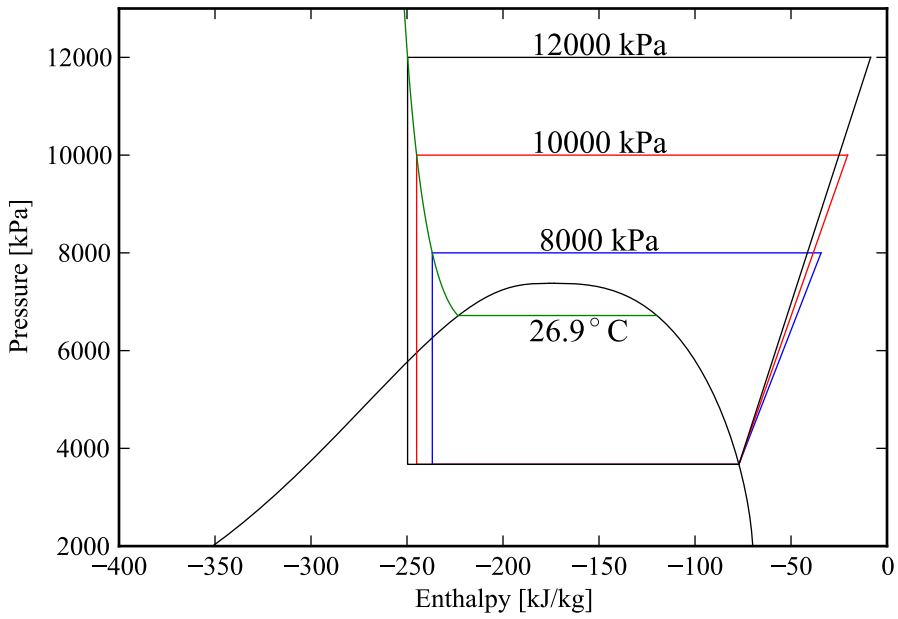


Figure 3.13. Pressure-Enthalpy plot for transcritical CO_2 cycles without flooding for varied gas cooler pressures with gas cooler outlet temperature fixed at 26.9°C.

clear that there is a oil mass fraction /gas cooler pressure that optimizes the flooded system COP, which in this case is an oil mass fraction of 0.376 and gas cooler pressure of 11400 kPa for a COP of 1.721. As the source and sink temperatures change, the shape of the COP surface will change as well, but in general an optimal pairing of oil mass fraction and gas cooler pressure can still be found.

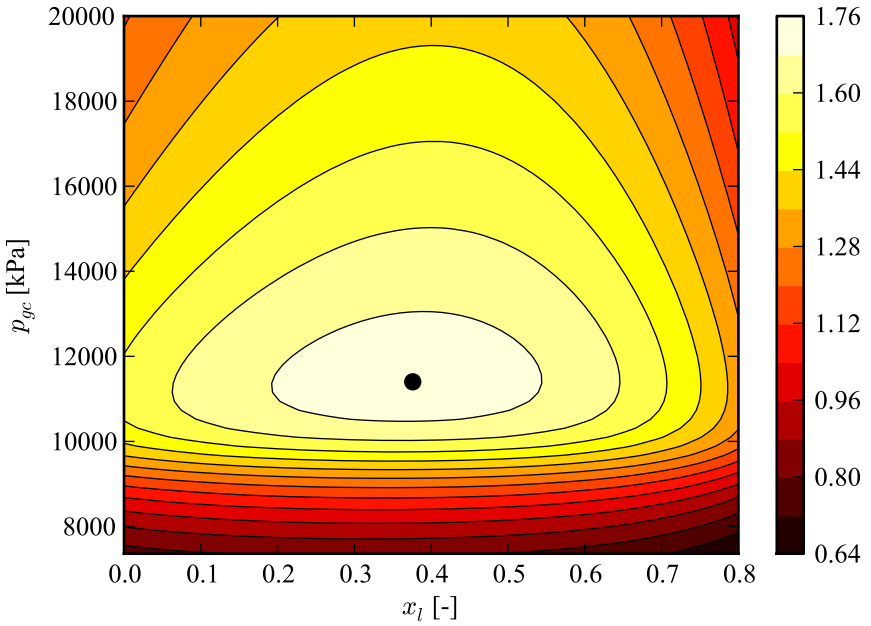
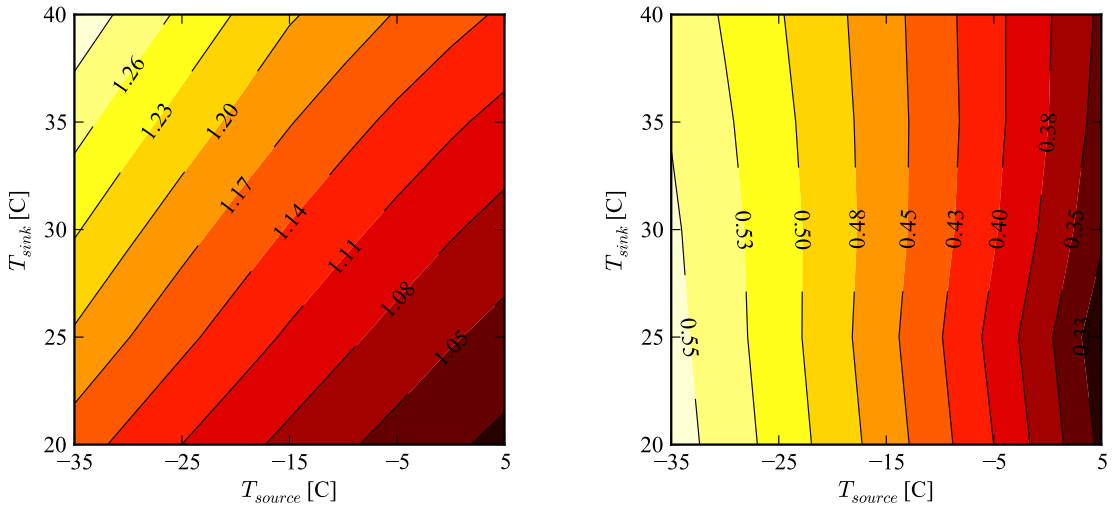


Figure 3.14. COP contours for a transcritical CO_2 system as a function of x_l and p_{gc} for T_{source}=5°C and T_{sink}=40°C.

If the flooded CO_2 cycle operates transcritically, it is then possible to optimize the cycle efficiency by modifying both the gas cooler pressure and the oil mass fraction. Figure 3.15 shows the results for a trancritical flooded system as a function of source and sink temperatures. As with the subcritical CO_2 flooded system, the greatest improvement is obtained for the largest temperature lift. In this case, the improvement in COP is over 31% for source and sink temperatures of -35°C and 40°C respectively. In addition, the optimal oil mass fraction has similar behavior to the subcritical CO_2 system. The optimal oil mass fraction is a weak function of the sink temperature but is a strong function of the source temperature.



(a) Contour plot of the ratio of COP of flooded systems to baseline systems as a function of the source and sink temperatures.

(b) Contour plot of the optimal oil mass fraction of flooded systems as a function of the source and sink temperatures.

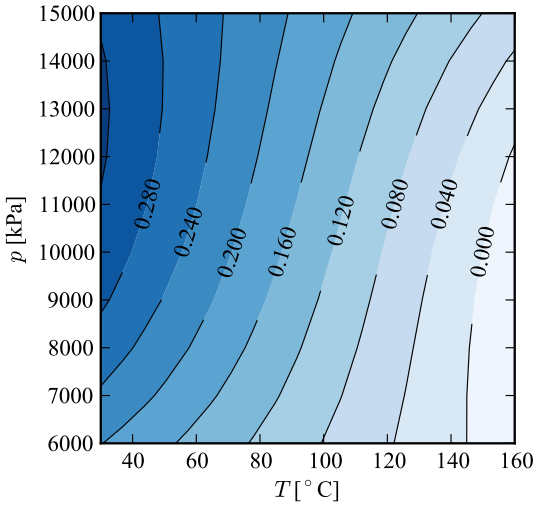
Figure 3.15. Transcritical CO_2 systems flooded with PAG oil.

The selection of flooding liquid for flooded compression is driven by a combination of the thermophysical properties of the flooding liquid as well as the solubility of the refrigerant in the flooding agent. The thermophysical properties of flooding agents are described above, and the solubility is considered here. The solubility is of importance in the separator because this is where the refrigerant streams split up. In the separator, some of the refrigerant gets solved in the flooding liquid and is cooled and throttled back to the compressor suction pressure without providing any cooling effect. This solved refrigerant both decreases the cooling capacity as well as increases the compression power required. The phase separator is effectively at the discharge temperature of the compressor (assuming no heat loss between compressor discharge port and phase separator).

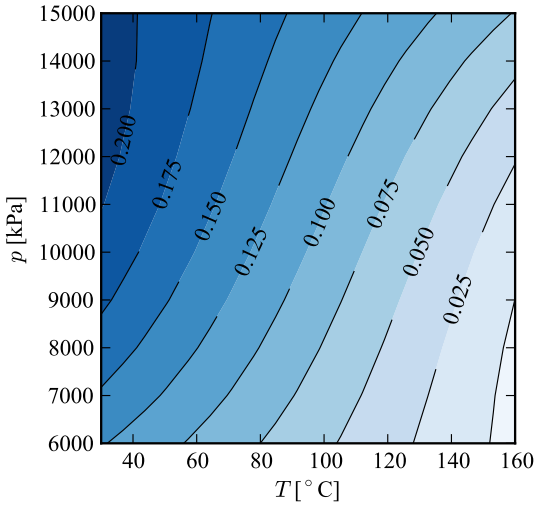
Solubility data for mixtures of refrigerants and flooding liquids are quite limited. For instance, Hauk (2000) provides some of the only experimental data available for the solubility of CO_2 in common refrigeration oils (PAO, POE, and PAG). The solu-

bility of CO_2 in water is better understood since it is a critical component of climatic modeling, and Duan (2003) presents exhaustive data for the solubility of CO_2 in water. In order to carry out the optimization of the flooded cycle with refrigerant solubility it is necessary to obtain curve fits for the solubility data. Duan provides solubility data over a sufficiently wide range of conditions so a polynomial regression for CO_2 solubility was developed as a function of pressure and temperature. On the other hand, the only two isotherms available from Hauk that are of use are the 40°C and 100°C isotherms. These are the isotherms where the gas still remains in the gas phase. The challenge for the CO_2/oil solubility data is that the discharge temperatures for CO_2 compressors can be above 100°C. For instance, compressing saturated vapor CO_2 isentropically from -20°C to 100 bar yields a compressor discharge temperature of 104.6°C. Therefore, correlations of the solubility fraction for CO_2 for temperatures of 40°C and 100°C as a function of pressure were developed, and interpolated or extrapolated as necessary for temperatures not equal to 40°C or 100°C. Figure 3.16 shows the results of this solubility analysis, and presents the solubility mass fractions of CO_2 for a wide range of temperature and pressure. Solubility fractions lower than 0.0 are impossible and are an artifact of extrapolation. Appendix A presents the correlations employed within the code for the flooded cycle modeling.

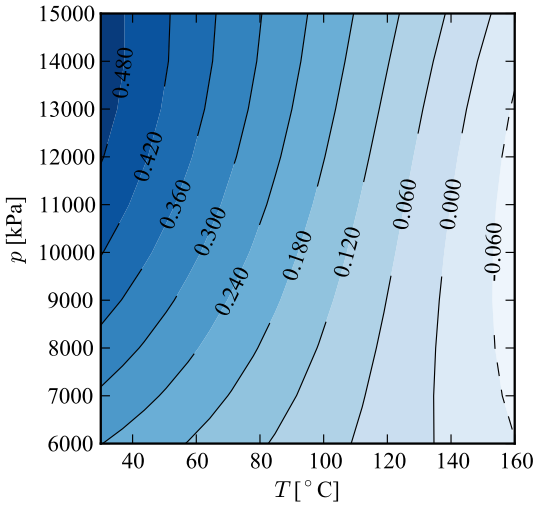
In general, as the pressure goes up, so does the solubility, and as the temperature goes up, the solubility decreases. The increase in solubility at higher pressures will tend to decrease the optimal oil mass fraction from the zero-solubility case due to the losses associated with solved refrigerant. The oils exhibit similar solubility characteristics, but the solubility of CO_2 in water is significantly lower than that of the oils.



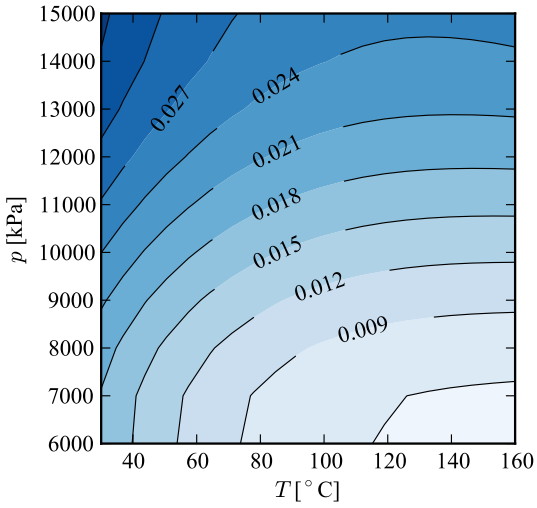
(a) CO_2 and PAG oil.



(b) CO_2 and PAO oil.



(c) CO_2 and POE oil.

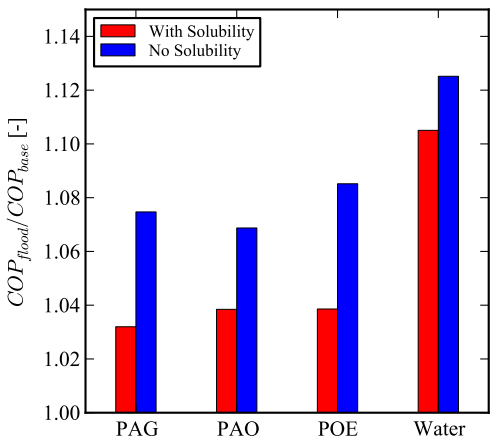


(d) CO_2 and Water.

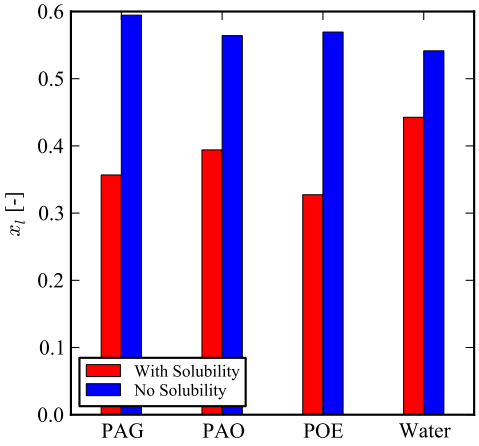
Figure 3.16. Contour plots of the solution mass fraction of CO_2 ($x_{g,s}$) in various fluids as a function of temperature and pressure. Based on the data of Hauk (2000) and Duan (2003).

Thus a cycle optimization can be carried out with the effects of refrigerant solubility included. Figure 3.17 presents the results for flooded CO_2 cycles with and without solubility included for a range of flooding agents for a high temperature lift application. This sink temperature is near the optimal intermediate temperature found by Getu (2008) for ammonia-CO_2 cascade cycles. With the addition of flooding (but without solubility effects), the increase in COP is greater than 7% for the oils and greater than 12% for water. The disparity in efficiency improvement between the oils and water is due to water's superior thermophysical properties for the flooding application. For instance, the specific heat of water is approximately 1.8 times higher than that of the highest density oil (POE). POE and PAG oil have specific gravities near those of water, so the differences in cycle performance are predominantly driven by the difference in specific heat rather than density. The optimal oil mass fraction is similar for all the flooding agents without solubility.

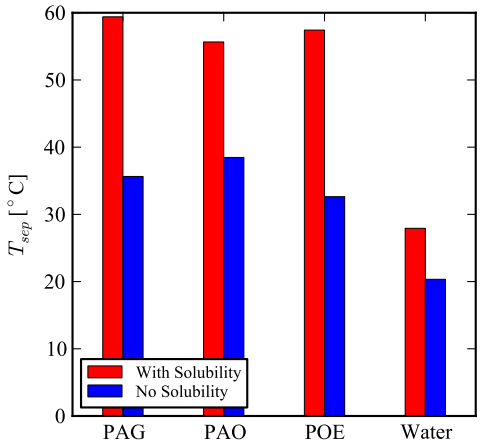
Once solubility effects are included, the benefits to cycle performance from the use of water become even clearer. The improvements in cycle efficiency for flooded compression are halved for the oils to around 3%, which the water-flooded system sees a relatively small decrease in its cycle performance due to the relatively low solubility of CO_2 in water. The optimal liquid mass fraction also decreases since the more oil that flows through the system, the more solved refrigerant will be transported through the oil loop. In addition there is a large disparity in the separation temperature between the oils and water. Again this is due to the extremely high specific heat of water which means that for the same amount of water, a larger decrease in discharge temperature is achieved. Also, the optimization for water is not strongly penalized by the increased solubility of CO_2 at lower temperatures like the oils, so a lower compressor discharge temperature (separator temperature) can be tolerated without greatly increasing the refrigerant solubility.



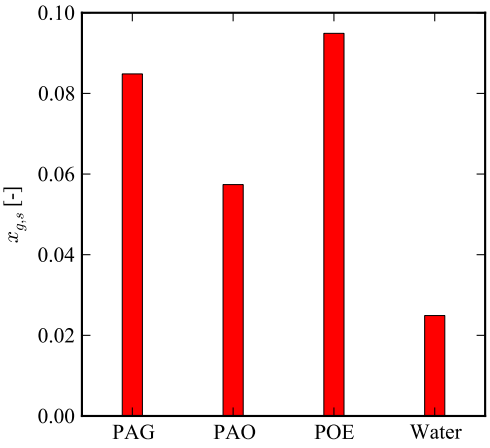
(a) Ratio of flooded to baseline COP



(b) Liquid mass fraction



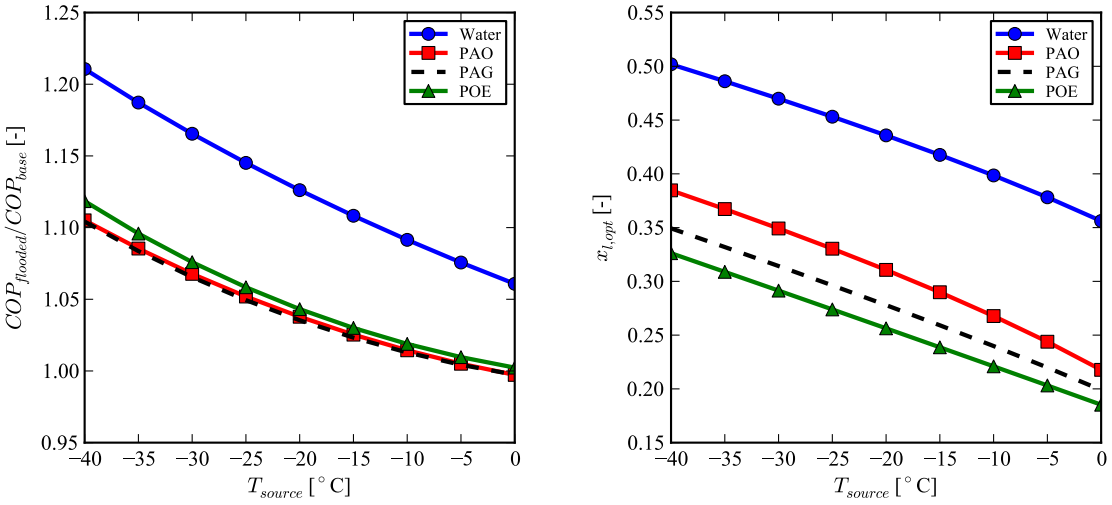
(c) Temperature of the separator



(d) Solved refrigerant mass fraction at the exit from separator

Figure 3.17. Optimal CO_2 cycle performance flooded with a range of flooding liquids, with and without solubility effects included for T_{source}=-40°C and T_{sink}=-10°C.

Finally the source temperature can be varied over a range of temperatures for all the flooding liquids while including the effects of solubility. Figure 3.18 presents the results for the flooded CO_2 system over a range of source temperatures including the effects of solubility. Water remains the best flooding agent from a thermodynamic standpoint.



(a) Ratio of flooded to baseline system COP as a function of source temperature.

(b) Optimal oil mass fraction as a function of source temperature.

Figure 3.18. Performance of flooded CO_2 system including the effects of solubility for T_{sink}=15°C.

The problem with using water as a flooding agent is two-fold. For one, the viscosity of water is significantly lower than that of the oils. For instance, the viscosity of water at 40°C is 0.653 mPa-s (Klein, 2010) while that of PAG oil 0-OB-1020 is 56.4 mPa-s (Booser, 1997). The very low viscosity of water means that water would struggle to maintain good lubrication in the compressor and could result in premature failure of compressor components.

The second problem with using water as a flooding agent is its chemical reactivity with CO_2. When CO_2 dissolves in water it forms carbonic acid, a weak acid. This acidic environment would likely cause corrosion problems for iron-based metal-

lic alloys, though the use of stainless steel could avoid the problems of corrosion. Unfortunately stainless steel has poor frictional characteristics.

3.4.3 Annual Energy Consumption Of Freeze Store

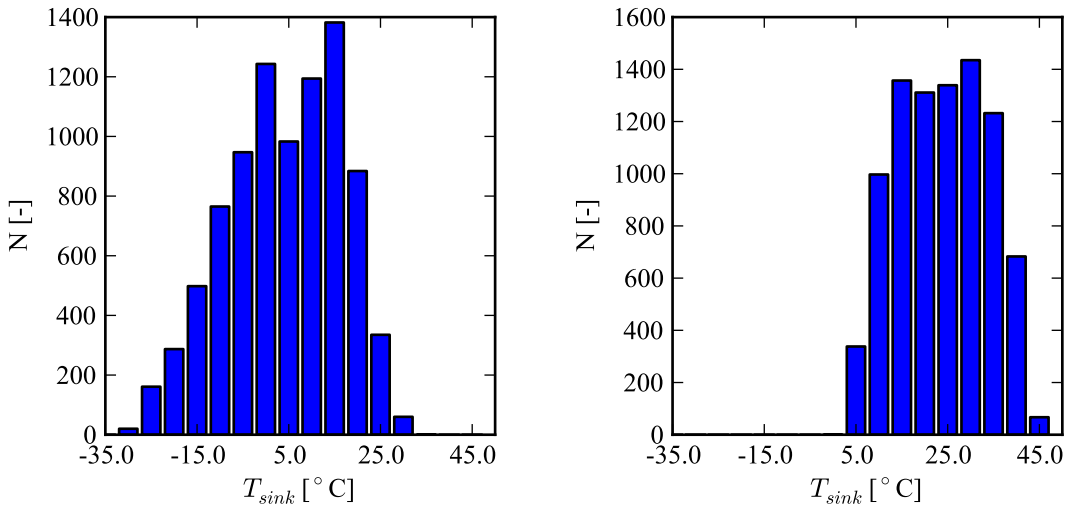
Figure 3.19. Map of cities used for TMY3 annual energy consumption comparison.

Up to this point, the analysis has been focused on theoretical systems with ficticious working temperatures. The analysis here is focused on a more practical application - the use of a refrigeration plant to maintain a 29,526 ft^2 cold store at -50°C for two different climates in the USA. The information required to calculate the refrigeration load is obtained from a worked example in the ASHRAE handbook (Garbarino, 2002) which yields the fixed loads and an overall conductance of the insulated envelope. From this problem, the fixed loads (including product, lighting, people, trucks, and fans) are equal to 155.9 kW and the overall conductance is found

to be 1.18 kW/K where all exterior surfaces of the thermal envelope are assumed to be at the ambient temperature. Thus the load of the freeze store is equal to

$$\dot{Q}_{load} = 155.9 \text{ kW} + (1.18 \text{ kW/K}) \cdot (T_{sink} - T_{source}). \tag{3.53}$$

The displacement of the compressor is scaled to yield the necessary capacity for each temperature bin. This assumes an ideal compressor with variable-frequency drive. Water is considered as the flooding agent. The flooded system operates subcritical when it can, and transcritical when it cannot. Where relevant, the oil mass flow fraction and gas cooler pressure are optimized.



(a) Duluth, Minnesota.

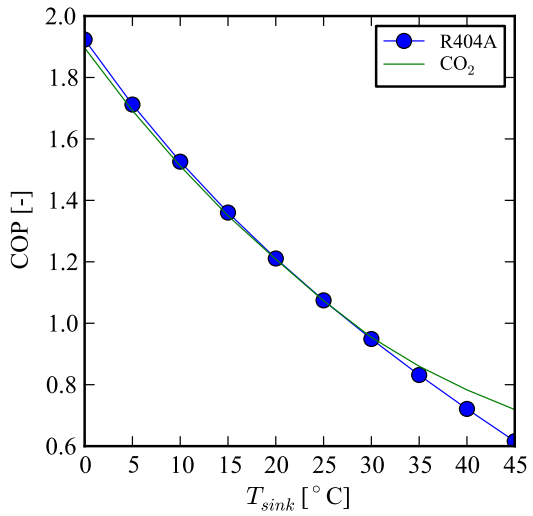
(b) Phoenix, Arizona.

Figure 3.20. Hourly Temperature Distribution based on TMY data.

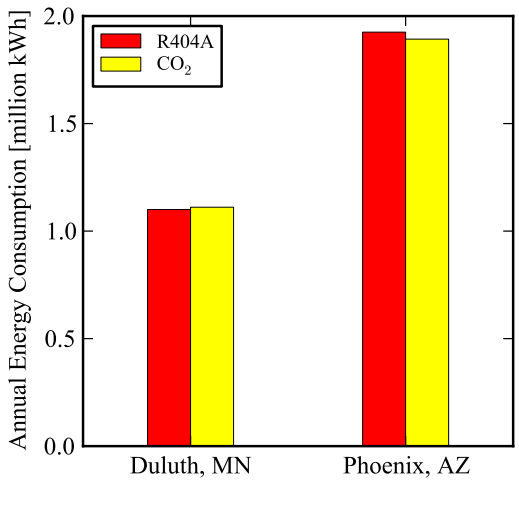
The geographic location of the freeze store has a large impact on the operating envelope of the refrigerating system due to the ambient temperature profile. For that reason two locations in the continental United States of America with vastly different climates were considered. The two cities are Duluth, Minnesota and Phoenix, Arizona, shown in Figure 3.19. These two cities span the climate spectrum from very cold to very hot and can give an idea about the performance of the water-flooded CO_2 system as compared with a baseline R404A system. The climate data

for the two locations are obtained from the typical meteorological year data (TMY3) compiled by the National Renewable Energy Laboratories (2005). Figure 3.20 shows the temperature distribution for both of the cities. Phoenix's temperature profile is weighted strongly towards high temperatures, with several hours per year above 40°C, with a high temperature of 44.4°C. On the opposite end of the climate spectrum is Duluth, with temperatures ranging from a minimum of -30.6°C to a maximum of 32.8°C. The temperatures for both Duluth and Phoenix are placed into bins that are 5°C wide. The model is then run for each binned ambient temperature.

The annual energy consumption of each freeze store can then be obtained by taking the sum of the products of each temperature bin's power consumption and the number of hours that fall into the temperature bin. The annual energy consumption is the objective function that any redesign or modification is trying to minimize.



(a) Freeze store COP versus sink temperature.



(b) Annual energy consumption of cold store at -40°C with TMY ambient air temperature data (R404A system baseline without flooding or regeneration, CO_2 system is flooded with water with regeneration).

Figure 3.21. TMY freeze store data.

Figure 3.21(a) shows the COP data from Phoenix for the flooded CO_2 system as well as the COP of the baseline R404A system that the CO_2 system is being compared to. Over a wide range of ambient temperatures, the water-flooded CO_2 system has competitive efficiency, and at higher ambient temperatures, the efficiency of the water-flooded CO_2 system is better than that of the R404A system.

Figure 3.21(b) shows the annual energy consumption for the freeze store for Duluth and Phoenix. The energy consumption of the freeze store operating with the refrigerant R404A is much higher in Phoenix than in Duluth due to the higher ambient temperatures. When the water-flooded CO_2 system is used to provide the cooling capacity for the freeze store located in Duluth, the result is a slight increase in the amount of electrical power required. On the other hand, when the freeze store in Phoenix uses an optimally water-flooded CO_2 refrigeration system, the energy consumption of the freeze store actually decreases. The important caveat here is that the R404A compressor is a single-stage compressor operating over a very large pressure ratio, for which typically two-stage compression with intercooling would be used. On the other hand, an adiabatic efficiency of the R404A compressor of 70% is used, which might be an optimistic target for such a large pressure lift. As was shown above, the benefit in efficiency for refrigerant R404A with oil flooding is quite large, but it seems possible that HFC refrigerants will be removed from the market by legislation in the near future. As a result, the results for refrigerant R404A with flooding were not considered here.

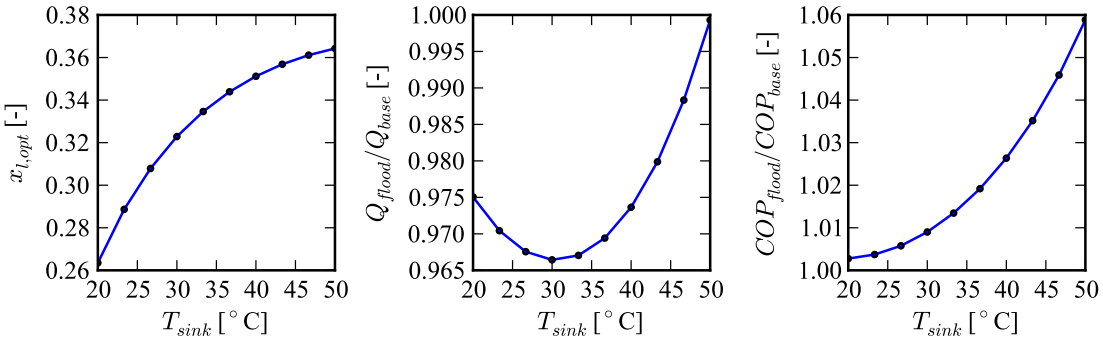
3.4.4 R410A Residential Heat Pump With Flooding And Regeneration

In the United States, the Air-Conditioning, Heating and Refrigeration Institute reports that more than 9 million residential unitary-type heat pumps were shipped in 2005 (the most recent year for which data is available), of which a large number operate with the refrigerant R410A (Air-Conditioning, Heating and Refrigeration Institute). Refrigerant R410A is also known by the trade name Puron. With such

a vast quantity of air-conditioning/heat-pump units produced there is an enormous potential for energy savings from even incremental improvements in system efficiency.

In this section, a heat pump operating with R410A will be investigated with the addition of oil flooding and regeneration. This study extends that of Hugenroth et al. (2006) which also investigated heat pumps operating with R410A though their study did not include the impact of regeneration.

As R410A is the working fluid, all state points of the cycle are subcritical. In addition, the indoor temperature selected for cooling mode is 23.88 °C (75°F). The compressor displacement rate is sized to yield approximately 10 kW cooling, or 3 tons of refrigeration, a standard capacity for residential heat pump units. At each point, the optimal oil mass flow fraction was found which optimizes the system COP, which is the heating COP for the heat pump and the cooling COP for the air-conditioner. The system parameters of Table 3.1 were used throughout, and PAG oil was used as the flooding agent.

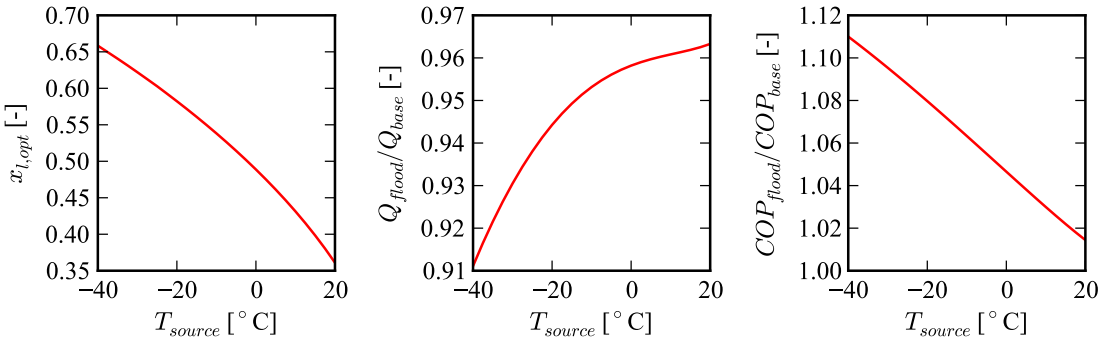


(a) Optimal oil mass fraction (b) Ratio of capacities (c) Ratio of COP

Figure 3.22. R410A air-conditioning unit with oil flooding.

Figure 3.22 shows the results for the R410A residential heat pump in cooling mode. From these results it is clear that the addition of oil flooding and regeneration is only marginally useful in cooling mode. The COP only increases 4.5% above that of the baseline system for very high ambient temperatures, while there is a small

net reduction in cooling capacity due to the volume that the oil takes in the suction chamber. Injecting the oil during the compression process after the suction chamber has closed might allow for a smaller reduction of capacity with oil flooding.

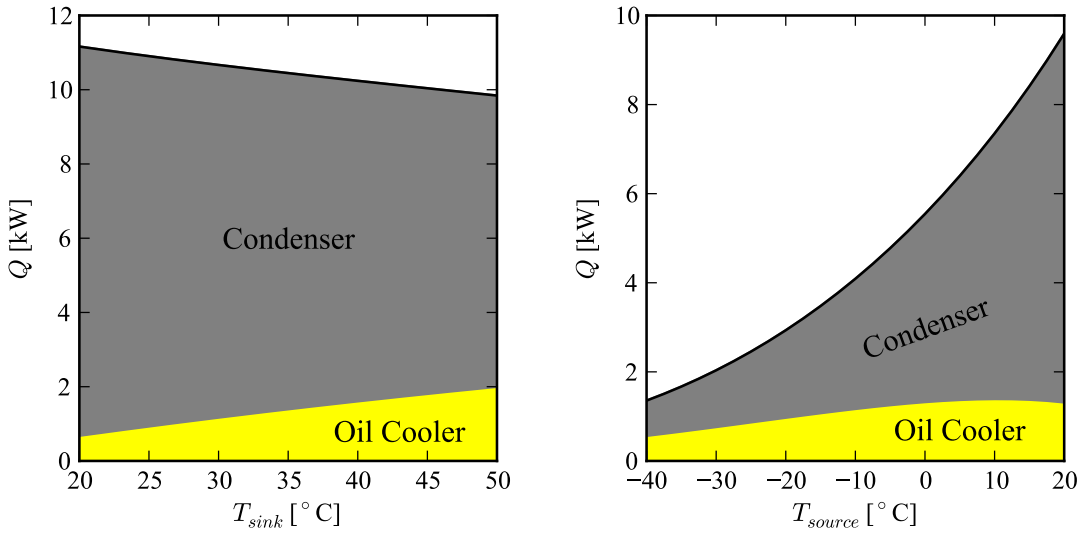


(a) Optimal oil mass fraction (b) Ratio of capacities (c) Ratio of COP

Figure 3.23. R410A heat pump with oil flooding.

Figure 3.23 shows the results for the heat pump in heating mode, which suggests that the benefits of oil cooling and regeneration are more strongly felt for heating mode operation at low outdoor temperatures. A condensing dew temperature of 43.3°C (110°F) is used. At outdoor temperatures down to -40°C, the increase in COP is up to 9.2%. In heating mode as well as cooling mode, the capacity decreases relative to the baseline system due to the decrease in available displacement volume caused by the presence of oil. At -40°C, the optimal oil mass fraction is 66%.

From a practical standpoint, the size of the heat rejection heat exchanger is of great interest. As more oil is injected, less of the heat rejection happens in the condenser part of the heat rejection heat exchanger and more happens in the oil cooler. Figure 3.24 shows the distribution of heat rejection between oil cooler and condenser for both cooling and heating modes for the reversible R410A heat pump. For this application, the percentage of heat rejection that occurs in the oil cooler is at most 41.8% at the lowest source temperatures in heating mode but otherwise is



(a) Cooling Mode

(b) Heating Mode

Figure 3.24. Distribution of heat rejection between oil cooler and condenser for R410A heat pump in heating and cooling modes with oil flooding.

a relatively small component of the heat rejection. This means that the oil cooler needed would be relatively small.

3.5 Summary

A simplified cycle model for flooded vapor compression has been proposed, for which the results show that CO_2 is a strong candidate, particularly when flooded with water. In order to realize this increase in system performance, compressors which can handle significant amounts of flooding liquid must be designed. The modeling presented in the subsequent chapters will provide the tools needed to analyze the flooded compression process, as well suggest design improvements for flooded compression.

CHAPTER 4. SCROLL COMPRESSOR GEOMETRIC MODEL

The modeling of scroll compressors is divided into two separate submodels. The first submodel handles the geometry of the compressor and calculates the chamber volumes, leakage areas, forces, etc.. The overall model is used to calculate the temperature, pressure, oil mass fraction, etc., throughout one rotation.

At a given step of the overall model, execution of the model requires first that the geometric model is run to generate the parameters which will be needed by the other submodels. In this way the submodels work in concert to arrive at the solution for a given set of input geometry and system state points.

The geometric model is composed of a number of sections which work together to calculate the necessary parameters for the scroll compressor. First, a short background of the underlying geometry of the scroll compressor is presented, and then this information is used to calculate the necessary output parameters.

4.1 Base Circle And Involutes

As has been demonstrated by Creux (1905), Yanagisawa (1990), Halm (1997), Wang (2005), Blunier (2009) and others, the geometry of the scroll compressor is quite complex. Fundamentally the geometry of the compressor is based on the form of an involute unwrapping from a circle. It is straight-forward to generate an involute with a bobbin of thread and a pen. With the pen attached to the end of the thread and held vertically, while maintaining the thread taut, unwrap the thread from the bobbin in a counter-clockwise fashion. The curve generated by the pen is an involute, just like in the scroll compressor. As the string begins to unwind counter-clockwise, the length of the unwound string increases linearly with the angle between the in-plane normal to the string and the horizontal axis, and is proportional to the radius

of the "spindle". This can be represented by the summation of two vectors, one which points from the origin to the edge of the "spindle", and the other from the spindle to the end of the "unwound string". This geometry is shown in Figure 4.1. From the above physical analogy, the vectors $\mathbf{r}_1$ and $\mathbf{r}_2$ from the origin to the base circle and from the base circle to a point on the involute respectively are given by

$$\begin{aligned} \mathbf{r}_1 &= r_b \cos\phi\hat{i} + r_b \sin\phi\hat{j} \\ \mathbf{r}_2 &= r_b \left(\phi - \phi_0\right) \sin\phi\hat{i} - r_b \left(\phi - \phi_0\right) \cos\phi\hat{j}. \end{aligned} \tag{4.1}$$

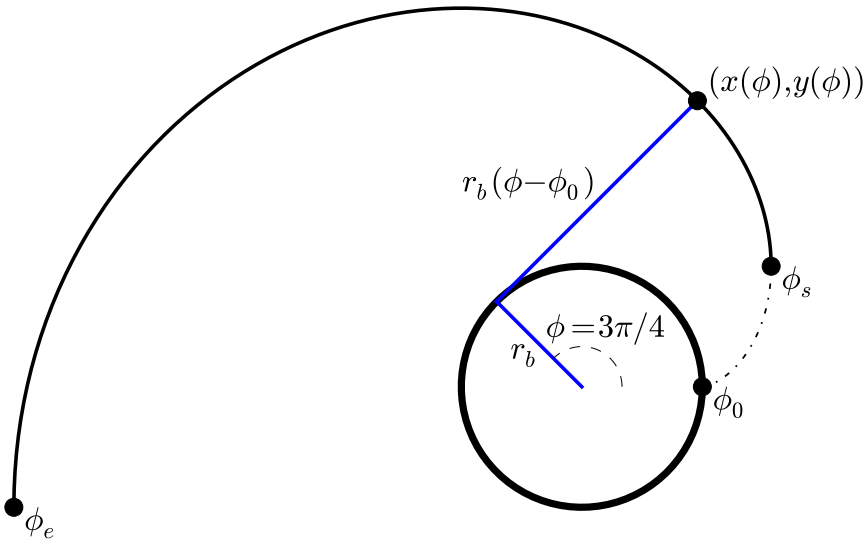


Figure 4.1. Involute Definition.

By fixing the origin of the coordinate system at the center of the base circle, the coordinates of a point on the involute with involute angle ϕ are given by

$$\begin{aligned} x &= r_b \left(\cos\phi + \left(\phi - \phi_0\right) \sin\phi\right) \\ y &= r_b \left(\sin\phi - \left(\phi - \phi_0\right) \cos\phi\right). \end{aligned} \tag{4.2}$$

For a given involute curve, ϕ takes all values in the range from ϕ_s to ϕ_e. The remaining part of the curve from ϕ_0 to ϕ_s does not form part of the scroll wrap because in that vicinity the scroll wrap is formed from arcs and lines to form the discharge geometry. The discharge chamber geometry will be covered in further detail in Section 4.9.

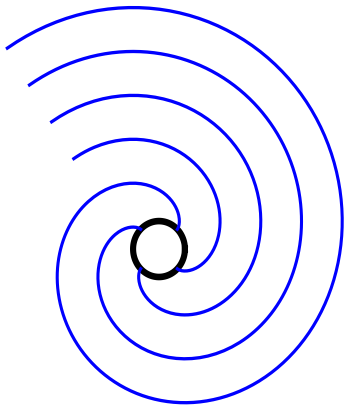


Figure 4.2. Involutes generated from base circle.

Selected members of the family of curves that can be generated from a given base circle are shown in Figure 4.2. A detailed view of the initial angles is shown in Figure 4.3. From this infinite set of involutes with varying initial angles, two are selected to form the inner and outer surfaces of one scroll wrap. By definition the outer surface is the surface that is further from the origin of the base circle. From a statics and material strength standpoint, one of the critical parameters is the thickness of the scroll. The scroll wrap thickness is typically on the order of 5 mm. The thickness of the scroll is given by

$$t_s = r_b \left(\phi_{i0} - \phi_{o0} \right). \tag{4.3}$$

The initial angles must be sorted and shifted by multiples of 2π such that $\phi_{i0} > \phi_{o0}$ (otherwise the thickness of the scroll wraps are negative based on the scroll thickness

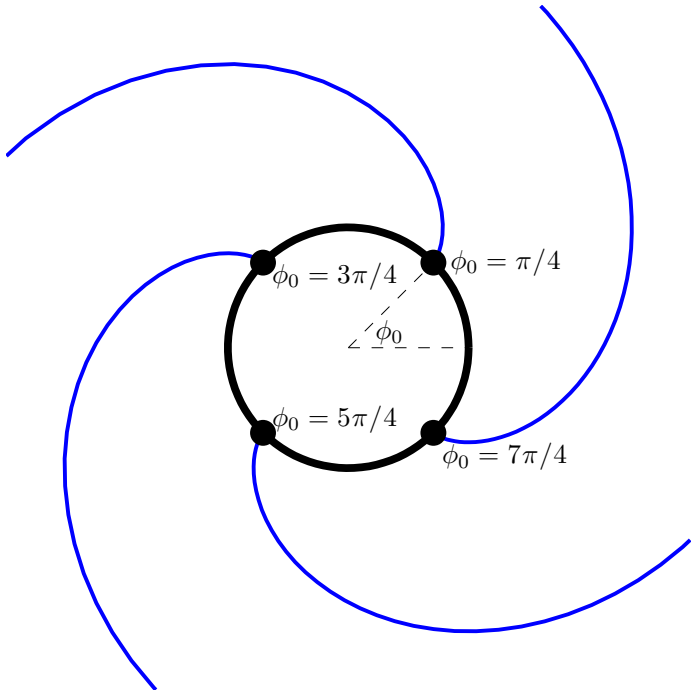


Figure 4.3. Detail of Initial Angles.

definition). Thus, after the initial angles have been selected for the fixed scroll, the coordinates of the points on the involutes of the fixed scroll are

$$\begin{aligned} x_{fi} &= r_b \left(\cos\phi + (\phi - \phi_{i0}) \sin\phi\right) \\ y_{fi} &= r_b \left(\sin\phi - (\phi - \phi_{i0}) \cos\phi\right) \\ x_{fo} &= r_b \left(\cos\phi + (\phi - \phi_{o0}) \sin\phi\right) \\ y_{fo} &= r_b \left(\sin\phi - (\phi - \phi_{o0}) \cos\phi\right) \end{aligned} \tag{4.4}$$

where the subscript fi corresponds to the inner involute of the fixed scroll, and fo to the outer involute of the fixed scroll.

4.2 Scroll Wraps

Once the geometry of the fixed scroll has been determined, that is, the involute angles for the inner and outer involutes have been determined, as shown in Figure 4.4, the fixed scroll is mated with the orbiting scroll. The geometry of the orbiting

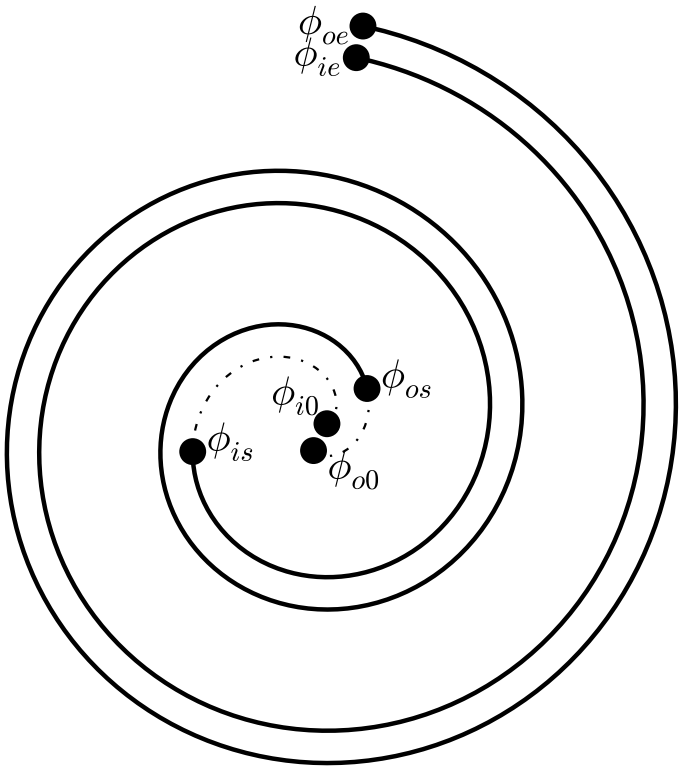


Figure 4.4. Involutes forming the fixed scroll and angle definitions.

scroll is exactly equivalent to that of the fixed scroll but reflected through the origin of the fixed scroll coordinate system. In this sense, the equivalence of the orbiting and fixed scrolls means that

$$\begin{aligned}
\phi_{i0} &= \phi_{fi0} = \phi_{oi0} \\
\phi_{is} &= \phi_{fis} = \phi_{ois} \\
\phi_{ie} &= \phi_{fie} = \phi_{oie} \\
\phi_{o0} &= \phi_{fo0} = \phi_{oo0} \\
\phi_{os} &= \phi_{fos} = \phi_{oos} \\
\phi_{oe} &= \phi_{foe} = \phi_{ooe}.
\end{aligned} \tag{4.5}$$

An important caveat here is that this pure symmetry only applies for scrolls that are fully symmetric, and some compressor manufacturers use scroll wraps that are not fully symmetric. In the case of offset scrolls, it is necessary to rederive all the scroll

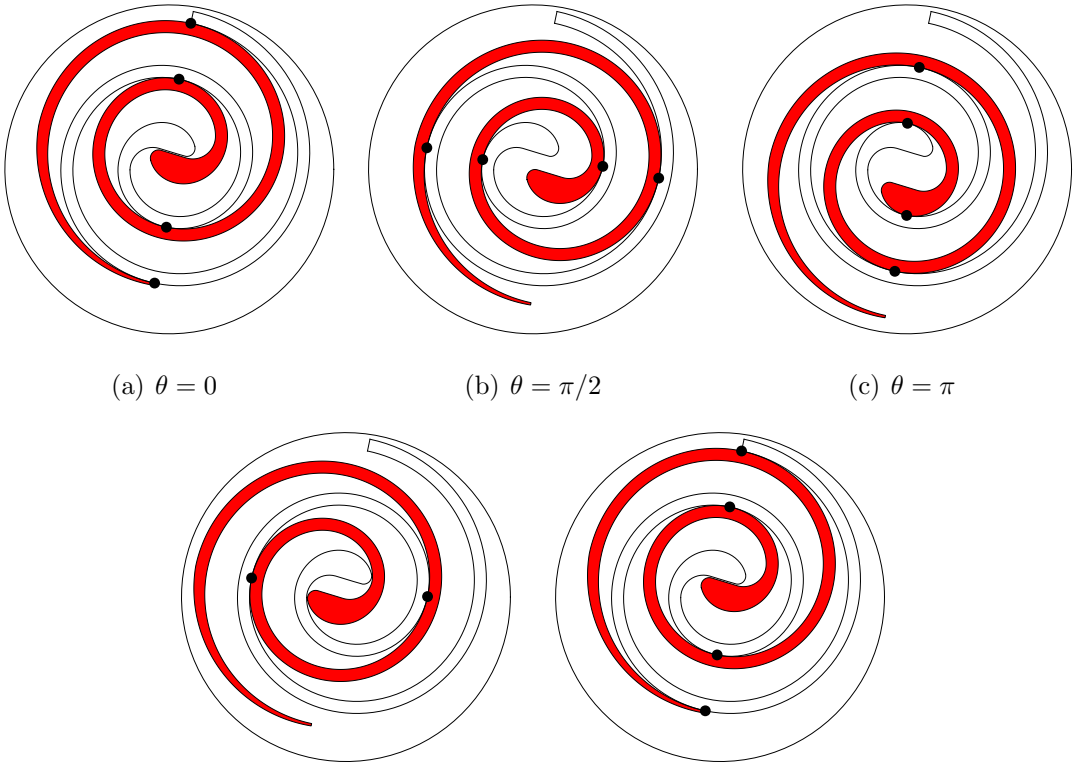
geometry. In the case that the scroll involute angles are not known *a priori*, it is possible to fit the involute angles to measures scroll wrap geometry, and a formal procedure is found in Appendix B.1. Several authors have proposed different definitions of the scroll involute angles, and the analysis required to convert the angles to the definitions employed here are found in Appendix B.2

The initial angles ϕ_{i0} and ϕ_{o0} determine the scroll geometry but are not part of the scroll involutes, and the same is true of the outer ending angles ϕ_{foe} and ϕ_{ooe}. These outer ending angles are generally set equal to the inner ending angles, but sometimes the outer parts of the scrolls are profiled or ground depending on the exact application. As the outer end surfaces of the scroll wraps do not form part of the working volume, the profiling or grinding will not impact the working volume. Therefore, using the same ending angle for inner and outer involutes for mathematical simplicity will not cause significant errors in volume prediction, even if the scrolls are ground. The subscripts beginning with "o" are are for the orbiting scroll, and those beginning with "f" are for the fixed scroll. For more information, refer to the nomenclature.

The orbiting scroll is then offset by a fixed amount away from the center of the base circle of the fixed scroll. Finally the orbiting scroll orbits (not rotates) around the center point of the base circle of the fixed scroll. The term orbiting is defined here as a motion where the Cartesian coordinate axes of the fixed scroll and orbiting scroll remain aligned while the central axes of the fixed and orbiting scrolls orbit around each other. In a real compressor, an Oldham ring is typically used to maintain the alignment of the scroll wraps, and an offset pin on the crankshaft is used to force the orbiting scroll to orbit at the orbiting radius.

Geometrically, the orbiting radius is calculated such that there is conjugacy between the orbiting and fixed scrolls. Mathematically, conjugacy means that the two scrolls are in perfect contact throughout the entire compression process and are tangent at a number of conjugacy points, as seen in Figure 4.5, where the conjugate points are shown for crank angles from $\theta = 0$ to $\theta = 2\pi$. The scrolls are in contact at four points at $\theta = \pi/2$, and at two points at $\theta = 3\pi/2$ since at $\theta = 3\pi/2$ the crank

angle is greater than the discharge angle and the innermost set of conjugate points has been swallowed into the discharge region.



(a) $\theta = 0$ (b) $\theta = \pi/2$ (c) $\theta = \pi$

(d) $\theta = 3\pi/2$ (e) $\theta = 2\pi$

Figure 4.5. Conjugate points over one rotation.

As the compression process proceeds, the conjugate points move towards the center (high pressure region) of the compressor. This can be seen by considering the first three frames of Figure 4.5. If the conjugate angle is based on the fixed inner involute, the farthest-out conjugate point begins the rotation at the inner ending involute angle, and after one rotation, the conjugate point has moved 2π radians towards the center of the compressor. Thus, the equation of the involute angles for all of the conjugate points on the inner involute of the fixed scroll is given by

$$\phi_{k,fi} = \phi_{fie} - \theta - (k-1)\,2\pi \tag{4.6}$$

where k =1, 2, ... is the index of the conjugate angle with k=1 corresponding to the index of the outermost conjugate point along the involute curve. The crank angle θ is defined such that the scrolls are tangent at the inner ending angle of the fixed scroll for $\theta = 0$. Positive values of θ correspond to compression, and θ takes on values in the range 0 to 2π inclusive. The case of $\theta = 0$ can be seen in Figure 4.6. The inner involute of the orbiting scroll mates with the outer involute of the fixed scroll. As a result, the conjugate angles (for $\phi_{fie} = \phi_{oie} = \phi_{foe} = \phi_{ooe}$) are calculated in a similar manner for the outer involute of the orbiting scroll, though shifted by π

$$\phi_{k,oo} = \phi_{fie} - \theta - (k-1)\,2\pi - \pi. \tag{4.7}$$

Similarly, the conjugate angles for the inner surface of the orbiting scroll and the outer surface of the fixed scroll are given by

$$\begin{aligned} \phi_{k,fo} &= \phi_{fie} - \theta - (k-1)\,2\pi - \pi \\ \phi_{k,oi} &= \phi_{fie} - \theta - (k-1)\,2\pi. \end{aligned} \tag{4.8}$$

For a given scroll wrap geometry, the required orbiting radius can be determined based on knowledge of the dimensions of the scrolls. As seen in Figure 4.6, there is a ray that is tangent to both base circles for orbiting and fixed scrolls, and also intersects the conjugate point at the end of the inner involute of the fixed scroll. The length of the line from A to D is given by the magnitude of the $\mathbf{r}_2$ vector ($r_b(\phi_{ie} - \phi_{i0})$ from Eqn. (4.1)) and the length of the line segment from B to C is calculated in the same manner. Thus, from the geometry presented above, it is possible to calculate the length of the line segment $\overline{AB}$ which is equal to the orbiting radius. The orbiting radius can be calculated as a function of only the inner ending angle, base circle radius, and initial angles from

$$\begin{aligned} r_o &= \left(\overline{AD} - \overline{BC}\right) - \overline{CD} \\ r_o &= \left(r_b\left[\phi_{ie} - \phi_{i0}\right] - r_b\left[(\phi_{ie} - \pi) - \phi_{i0}\right]\right) - t_s \\ r_o &= r_b\pi - t_s = r_b(\pi - \phi_{i0} + \phi_{o0}). \end{aligned} \tag{4.9}$$

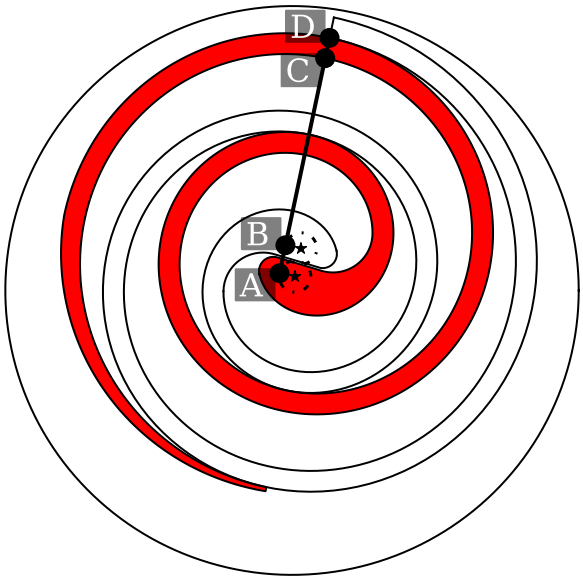


Figure 4.6. Geometry for calculation of r_o (θ=0).

For plotting and other analysis, it is useful to know the Cartesian coordinates of the points along the orbiting scroll wraps. To enable this capability, the crank angle needs to be shifted. The coordinates of the involutes forming the orbiting scroll with the offset angle θ_Δ are given by

$$\begin{aligned} x_{oi} &= -r_b\left(\cos\phi + (\phi - \phi_{i0})\sin\phi\right) + r_o\cos\left(\theta_\Delta - \theta\right) \\ y_{oi} &= -r_b\left(\sin\phi - (\phi - \phi_{i0})\cos\phi\right) + r_o\sin\left(\theta_\Delta - \theta\right) \\ x_{oo} &= -r_b\left(\cos\phi + (\phi - \phi_{o0})\sin\phi\right) + r_o\cos\left(\theta_\Delta - \theta\right) \\ y_{oo} &= -r_b\left(\sin\phi - (\phi - \phi_{o0})\cos\phi\right) + r_o\sin\left(\theta_\Delta - \theta\right) \end{aligned} \tag{4.10}$$

where the parameter θ_Δ is not yet known. This shift of the orbiting angle is required because the definition that θ is equal to zero at the beginning of the suction process is not consistent with the outer-most conjugate point being located at the end of the inner fixed scroll for $\theta = 0$. To calculate the required shift, the scrolls are assumed to be conjugate at the end of the inner involute of the fixed scroll. The coordinates of the point D of Figure 4.6 on the outer involute of the orbiting scroll and the inner

involute of the fixed scroll are then equated. Equating the coordinates of this point yields

$$\begin{aligned} x: & \begin{cases} r_b\left(\cos\left(\phi_e-\theta\right)+\left(\phi_e-\theta-\phi_{i0}\right)\sin\left(\phi_e-\theta\right)\right) \\ =-r_b\left(\cos\left(\phi_e-\theta-\pi\right)+\left(\phi_e-\theta-\pi-\phi_{o0}\right)\sin\left(\phi_e-\theta-\pi\right)\right)+r_o\cos\left(\theta_\Delta-\theta\right) \end{cases} \\ y: & \begin{cases} r_b\left(\sin\left(\phi_e-\theta\right)-\left(\phi_e-\theta-\phi_{i0}\right)\cos\left(\phi_e-\theta\right)\right) \\ =-r_b\left(\sin\left(\phi_e-\theta-\pi\right)-\left(\phi_e-\theta-\pi-\phi_{o0}\right)\cos\left(\phi_e-\theta-\pi\right)\right)+r_o\sin\left(\theta_\Delta-\theta\right) \end{cases} \end{aligned} . \tag{4.11}$$

Setting θ equal to zero and solving for θ_Δ yields

$$\theta_\Delta=\phi_{ie}-\frac{\pi}{2}. \tag{4.12}$$

Thus, the coordinates of the involutes for the orbiting scroll are given by

$$\begin{aligned} x_{oi} &= -r_b\left(\cos\phi+\left(\phi-\phi_{i0}\right)\sin\phi\right)+r_o\cos\left(\phi_{ie}-\tfrac{\pi}{2}-\theta\right) \\ y_{oi} &= -r_b\left(\sin\phi-\left(\phi-\phi_{i0}\right)\cos\phi\right)+r_o\sin\left(\phi_{ie}-\tfrac{\pi}{2}-\theta\right) \\ x_{oo} &= -r_b\left(\cos\phi+\left(\phi-\phi_{o0}\right)\sin\phi\right)+r_o\cos\left(\phi_{ie}-\tfrac{\pi}{2}-\theta\right) \\ y_{oo} &= -r_b\left(\sin\phi-\left(\phi-\phi_{o0}\right)\cos\phi\right)+r_o\sin\left(\phi_{ie}-\tfrac{\pi}{2}-\theta\right). \end{aligned} \tag{4.13}$$

With reference to Eqn. (4.4), it is clear that the orbiting scroll coordinates are just the fixed scroll coordinates mirrored through the origin and offset by the orbiting radius. The term

$$\theta_m=\phi_{ie}-\frac{\pi}{2}-\theta \tag{4.14}$$

is defined for simplicity.

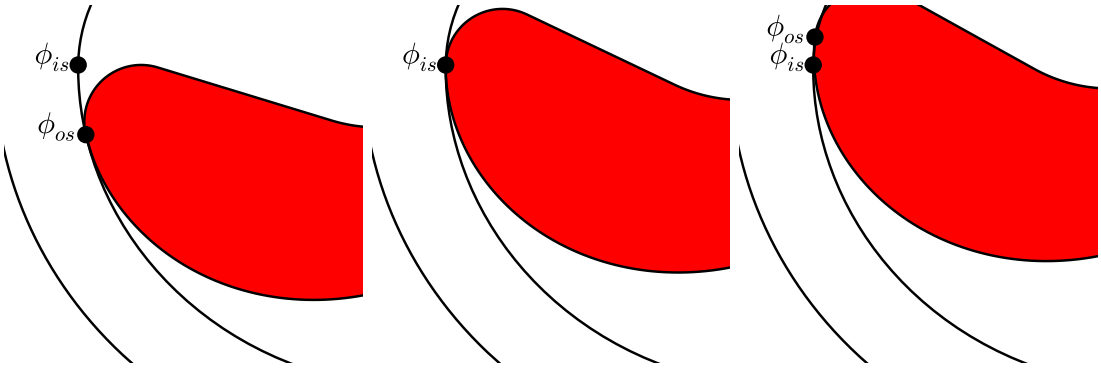
4.3 Discharge Angle

At the discharge angle, the compression chambers open up to the discharge region. There are two possible scenarios which govern the angle at which the discharge process begins:

- If $\phi_{os}>\phi_{is}-\pi$, the inner-most conjugate point on the outer involute arrives at the outer involute starting angle before the inner-most conjugate point on the

inner involute arrives at the inner involute starting angle. Figure 4.7(a) shows this case.

- If $\phi_{os} = \phi_{is} - \pi$, both inner-most inner and outer involute conjugate points arrive at the respective starting angles simultaneously. Figure 4.7(b) shows this case.



(a) $\phi_{os} > \phi_{is} - \pi$ (b) $\phi_{os} = \phi_{is} - \pi$ (c) $\phi_{os} < \phi_{is} - \pi$ (Collision)

Figure 4.7. Detail of discharge region for $\theta = \theta_d$.

It is not possible that $\phi_{os} < \phi_{is} - \pi$, for if this were the case, the involute portions of the two scrolls would collide at the discharge angle, as seen in Figure 4.7(c). The condition that $\phi_{os} \geq \phi_{is} - \pi$ is a necessary but not sufficient condition to ensure that no collision occurs, as a poorly selected discharge geometry can still result in collision. The angle at which the discharge process begins can be determined by finding the crank angle at which the innermost conjugate point on the outer involute has reached the starting angle of the outer involute. The conjugate angle on the outer involute of the orbiting scroll is given by Eqn. (4.7), with $N_c = k - 1$ since $k = 2$ for the outer-most pair of compression chambers. This means that

$$\phi_{os} = \phi_{ie} - \theta - 2\pi N_c - \pi \tag{4.15}$$

where N_c is the number of pairs of compression chambers in existence at the crank angle θ. The maximum number of compression chambers occurs at the beginning of the rotation (when $\theta = 0$), and is given by

$$N_{c,max} = floor\left(\frac{\phi_{ie} - \phi_{os} - \pi}{2\pi}\right) \tag{4.16}$$

where the *floor()* function rounds down the argument to the nearest integer. Thus, the discharge angle of the compressor can determined from

$$\theta_d = \phi_{ie} - \phi_{os} - 2\pi N_{c,max} - \pi. \tag{4.17}$$

4.4 Chamber Definitions

As seen in Figure 4.8, the definitions of the compressor control volumes change throughout the compression process. There are two parallel compression paths, which are given indices 1 and 2, where the index 1 is given to the chamber which is formed with the fixed scroll as its outer surface. Therefore, the suction chamber of the first path is named s_1. At the beginning of the rotation, the merged discharge region *ddd* is at a uniform pressure, and the compression chambers c_1 and c_2 are at the final pressure of the suction chambers from the previous rotation. When only one set of compression chambers exist, the compression chamber pair index is dropped for clarity. As the rotation proceeds, the suction chambers s_1 and s_2 open up and grow to have a finite volume. The compression chambers begin to compress the gas, and the suction chambers continue to draw in suction gas while the discharge chamber discharges the gas at high pressure. Right after the discharge angle, the compression chambers open up to the discharge region and are now referred to as discharge chambers d_1 and d_2 , and the chamber connected directly to the discharge port is referred to as *dd*. Fairly quickly the discharge chambers d_1 and d_2 and the discharge port chamber *dd* equalize in pressure, at which point d_1, d_2 and *dd* chambers are merged back into the grouped discharge chamber *ddd*. The rotation continues to the end, at which point the suction chambers become compression chambers, the

suction chamber goes back to having an infinitely small volume, and the compression process repeats. In addition, there is a suction channel which the gas enters when it comes into the compressor, which is given the name *sa*.

4.5 Mathematical Interlude

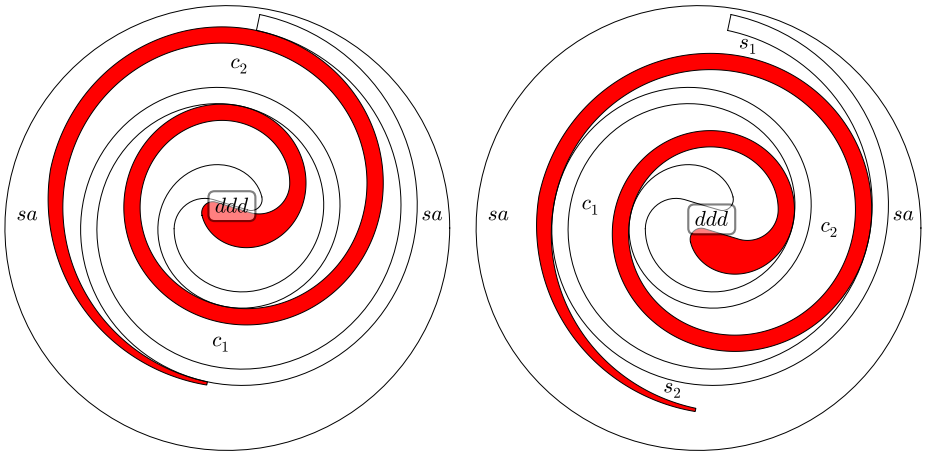
Before the compressor geometry can be analyzed, some mathematical tools need to be introduced in order to carry out the necessary calculations. The purpose of the next few sections is to provide exact analytic models (or very good approximations to the analytic models when exact analytic solutions are not available) for the geometry of the chambers of the compressor. The parameters to be calculated are

- Volumes of all chambers as a function of the crank angle
- Derivatives of the volumes of all chambers with respect to the crank angle
- Centroid of each chamber (needed to calculate the overturning moments due to the gas forces on the orbiting scroll)
- In-plane force components on the orbiting scroll due to pressure forces
- Radial leakage areas between chambers

Ultimately all these parameters will be integrated into the overall model. The use of analytic solutions avoids the imprecision of numerical approximations and numerical derivatives and provides solutions which are smooth, both factors which improve the ease of solution of the overall model. Numerical solutions based on high-accuracy polygons are also presented for completeness.

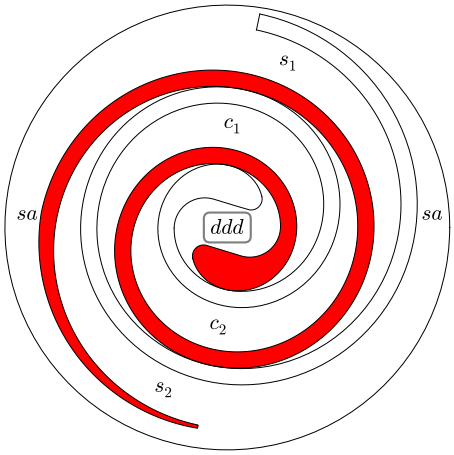
4.5.1 Analytic Area Calculations

The volume calculations required for the compressor are based on two types of area elements - triangles and parametrized curves. With these two elements, all the geometry of the scroll compressor can be analyzed.

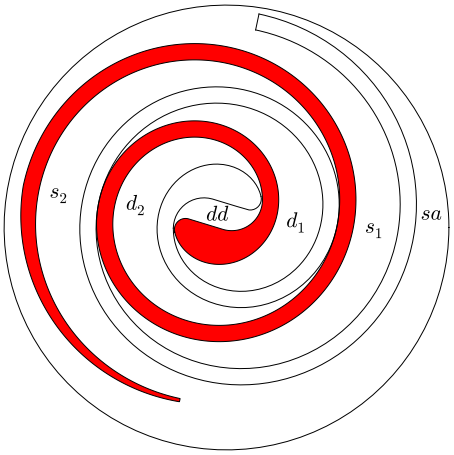


(a) $\theta = 0$

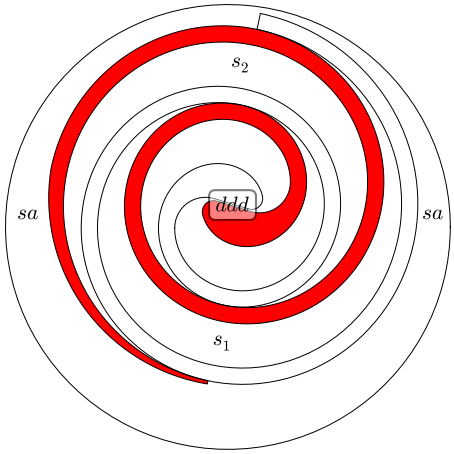
(b) $\theta = \pi/2$



(c) $\theta = \pi$



(d) $\theta = 3\pi/2$



(e) $\theta = 2\pi$

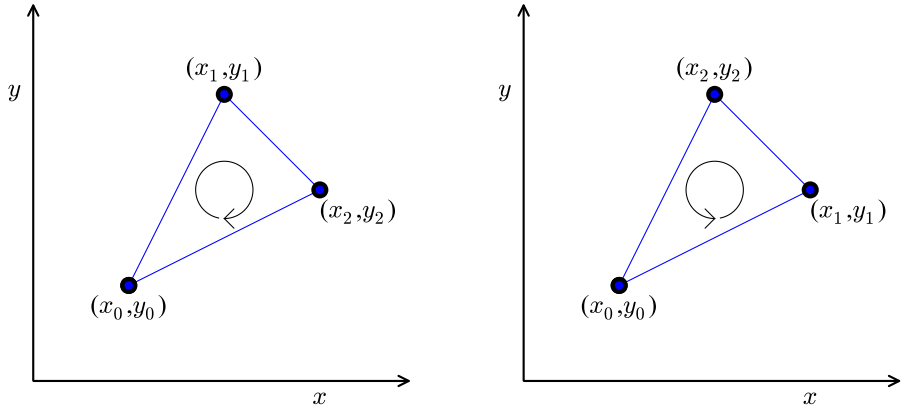
Figure 4.8. Compressor chamber definitions over one rotation.

Triangle

The solution for the triangle has a simple analytic form. For a triangle with vertices $(x_0,\, y_0)$, $(x_1,\, y_1)$, and $(x_2,\, y_2)$, the area of the triangle is simply

$$A = \frac{1}{2}\left[(x_1 - x_0)(y_2 - y_0) - (y_1 - y_0)(x_2 - x_0)\right]. \tag{4.18}$$

where the area of A can be either positive or negative depending on the order of the vertices. Figure 4.9 shows triangles with their vertices in clockwise and counter-clockwise orders, where the index i for the point $(x_i,\, y_i)$ goes from 0 to 2. If the vertices are in clockwise order, the area of the triangle is defined to be negative, or if ordered counter-clockwise, the area is positive. This form for the area arises by taking one half the cross-product of vectors from $(x_0,\, y_0)$ to $(x_1,\, y_1)$ and $(x_0,\, y_0)$, to $(x_2,\, y_2)$, respectively.



(a) Vertices oriented clockwise (b) Vertices oriented counter-clockwise

Figure 4.9. Triangles with clockwise and counter-clockwise orientations.

In the analysis which follows, the majority of the triangular area elements are selected in order to have one vertex at the origin of the coordinate system so that

the coordinates of one of the vertices drop out. If (x_0, y_0) is placed at the origin with coordinates $(0, 0)$, the area is then

$$A = \frac{1}{2}\left[x_1 y_2 - y_1 x_2\right]. \tag{4.19}$$

The centroid of the triangle is equal to the average of the Cartesian coordinates of the vertices, expressed mathematically as

$$c_x = \frac{x_0 + x_1 + x_2}{3} \tag{4.20}$$

$$c_y = \frac{y_0 + y_1 + y_2}{3}. \tag{4.21}$$

The centroid of the triangle is also the intersection of the set of lines between each vertex and the midpoint of the edge opposite, as seen in Figure 4.10

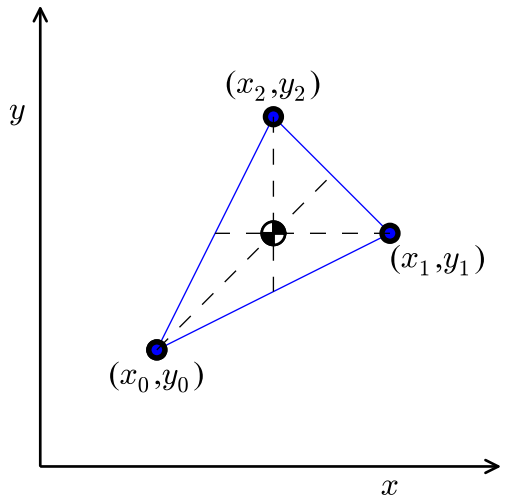


Figure 4.10. Centroid of triangle.

Parametrized curve

For the parametrized curve, the analysis proceeds in a similar fashion. A general parametric curve has the coordinates $(x(\phi), y(\phi))$ where ϕ is the parameter defining

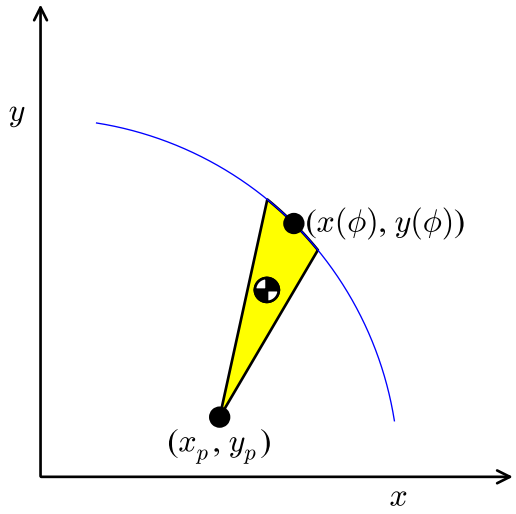


Figure 4.11. Definition of the differential area for a general parametric curve.

the curve. Figure 4.11 shows one parametric curve (an arc of a circle). The differential area between the curve and a point (x_p, y_p) is integrated to determine the total area between a point and the curve. The shaded differential area is equal to (Kreyszig, 2006)

$$dA = \frac{1}{2}\left[(x - x_p)\frac{dy}{d\phi} - (y - y_p)\frac{dx}{d\phi}\right] d\phi \tag{4.22}$$

which is one-half the cross-product of a vector from the (x_p, y_p) to a point on the curve and the tangent vector at that point on the curve which is analogous to the triangle analysis. The total area between a parametric curve and a point is therefore

$$A = \frac{1}{2}\int_{\phi_1}^{\phi_2}\left[(x - x_p)\frac{dy}{d\phi} - (y - y_p)\frac{dx}{d\phi}\right] d\phi \tag{4.23}$$

where ϕ_1 and ϕ_2 are selected such that the curve is traversed in a counter-clockwise direction relative to the point around which the integration is carried out in order to yield a positive differential area by the right-hand rule[1], where the first vector for the

[1]In order to apply the right hand rule, point the fingers of right hand along the direction of the first vector of the cross product, and wrap the fingers of the right hand in the direction of the second vector. If the thumb points upwards, the cross-product is positive.

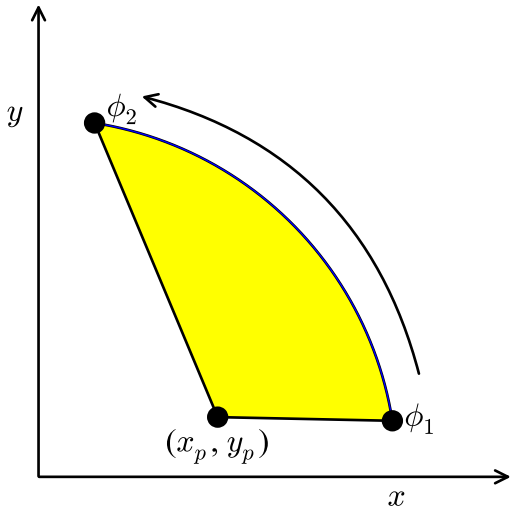


Figure 4.12. Area between a point and parametric curve traversed in the counter-clockwise direction.

right hand rule goes from (x_p, y_p) to a point on the curve. This integration can be seen in Figure 4.12.

The calculation of the centroid is obtained from

$$c_x = \frac{\int x_c dA}{\int dA} \tag{4.24}$$

$$c_y = \frac{\int y_c dA}{\int dA} \tag{4.25}$$

where x_c and y_c are the Cartesian coordinates of the centroid of the differential area element, shown with the center of mass marker in Figure 4.11. As the differential area element is locally an arc of a circle, the location of the centroid of the differential area is two-thirds of the way from (x_p, y_p) to the point on the curve. This relationship can be derived by beginning with the centroid of a circular sector (Gere, 2001) oriented along the x-axis (and shown in Figure 4.13), and taking the limit as the swept angle goes to zero, given by

$$x_c = \lim_{\gamma \to 0} \frac{2r \sin \gamma}{3\gamma} = \frac{2r}{3} \tag{4.26}$$

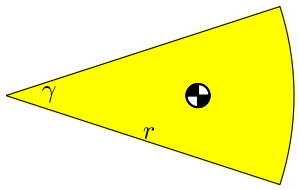


Figure 4.13. Circular sector.

with one application of L'Hôpital's rule. Thus, any differential area arc will have its centroid two-thirds of the way from its root to the radius of the arc. Substituting the location of the centroid of the differential area back into Eqn. (4.24) yields

$$c_x = \frac{\int_{\phi_1}^{\phi_2} \left(\frac{2}{3}x + \frac{1}{3}x_p\right) dA}{\int_{\phi_1}^{\phi_2} dA} \tag{4.27}$$

$$c_y = \frac{\int_{\phi_1}^{\phi_2} \left(\frac{2}{3}y + \frac{1}{3}y_p\right) dA}{\int_{\phi_1}^{\phi_2} dA}. \tag{4.28}$$

In order to calculate the pressure force components from the gas, the unit normal vectors are needed for both the inner and outer involutes of the orbiting scroll. The components of the unit normal vector for a general parametrized curve can be obtained from

$$\mathbf{n}_x = \frac{\left(\frac{dy}{d\phi}\right)}{\sqrt{\left(\frac{dx}{d\phi}\right)^2 + \left(\frac{dy}{d\phi}\right)^2}} \tag{4.29}$$

$$\mathbf{n}_y = \frac{-\left(\frac{dx}{d\phi}\right)}{\sqrt{\left(\frac{dx}{d\phi}\right)^2 + \left(\frac{dy}{d\phi}\right)^2}}. \tag{4.30}$$

There are two unit normal vectors for a given point on the curve and the other unit normal vector can be obtained by multiplying the obtained unit normal vector by -1 which flips the direction of the unit normal vector. For the involutes of the orbiting scroll, the direction of the unit normal vector is selected such that the unit normal vector points towards the scroll wrap since this is the direction that the force of the

pressure acts. Thus, the unit normal vectors for the inner and outer involutes of the orbiting scroll are given by

$$\mathbf{n}_{oi} = -\sin(\phi)\hat{i} + \cos(\phi)\hat{j} \tag{4.31}$$

$$\mathbf{n}_{oo} = \sin(\phi)\hat{i} - \cos(\phi)\hat{j} \tag{4.32}$$

and for the sake of completeness, the unit normal vectors of the inner and outer involutes for the fixed scroll are

$$\mathbf{n}_{fi} = \sin(\phi)\hat{i} - \cos(\phi)\hat{j} \tag{4.33}$$

$$\mathbf{n}_{fo} = -\sin(\phi)\hat{i} + \cos(\phi)\hat{j}. \tag{4.34}$$

Since the direction of the unit normal vectors have been selected to point towards the orbiting scroll wrap and the pressure and normal vectors are coincident, the differential of force vectors on the orbiting scroll are given by

$$d\mathbf{F}_{oi} = -pdA\sin(\phi)\hat{i} + pdA\cos(\phi)\hat{j} \tag{4.35}$$

$$d\mathbf{F}_{oo} = pdA\sin(\phi)\hat{i} - pdA\cos(\phi)\hat{j} \tag{4.36}$$

where the differential area over which the force is applied is equal to

$$dA = h_s r_b(\phi - \phi_0) \tag{4.37}$$

which yields

$$d\mathbf{F}_{oi} = -ph_s r_b\left[(\phi - \phi_{i0})\sin(\phi)\hat{i} - (\phi - \phi_{i0})\cos(\phi)\hat{j}\right] \tag{4.38}$$

$$d\mathbf{F}_{oo} = ph_s r_b\left[(\phi - \phi_{o0})\sin(\phi)\hat{i} - (\phi - \phi_{o0})\cos(\phi)\hat{j}\right]. \tag{4.39}$$

The coordinates of the pin on the shaft that drives the compression process are given by

$$x_{\bigcirc} = r_o\cos(\phi_{ie} - \pi/2 - \theta) \tag{4.40}$$

$$y_{\bigcirc} = r_o\sin(\phi_{ie} - \pi/2 - \theta). \tag{4.41}$$

The moments around the crank pin need to be overcome by the coupling mechanism that enforces the orbiting motion. There is also an overturning force from the fact

that the forces are not applied inline with the shaft pin, but this overturning force is neglected. Thus the moments from the orbiting scroll involutes are equal to

$$d\mathbf{M}_{\bigcirc,oi} = \mathbf{r}_{\bigcirc,oi} \times d\mathbf{F}_{oi} \tag{4.42}$$

$$d\mathbf{M}_{\bigcirc,oo} = \mathbf{r}_{\bigcirc,oo} \times d\mathbf{F}_{oo} \tag{4.43}$$

where $\mathbf{r}_{\bigcirc}$ is a vector from ($x_{\bigcirc}$, $y_{\bigcirc}$) to a point on the respective involute

Polygons

For the purposes of validation of the mathematical solutions of the analytic volume relations as well as to permit the analysis of more complex geometry that is not amenable to analytic representation, irregular polygons can be found which define the outer boundary of an area of interest. For a polygon defined with *n+1* vertices, the area of the polygon can be found from

$$A = \frac{1}{2}\sum_{i=0}^{n-1}\left(x_i y_{i+1} - x_{i+1} y_i\right) \tag{4.44}$$

where the polygon must be closed, or (x_0, y_0)=(x_n, y_n). Finally the centroid of the enclosed area can be obtained from

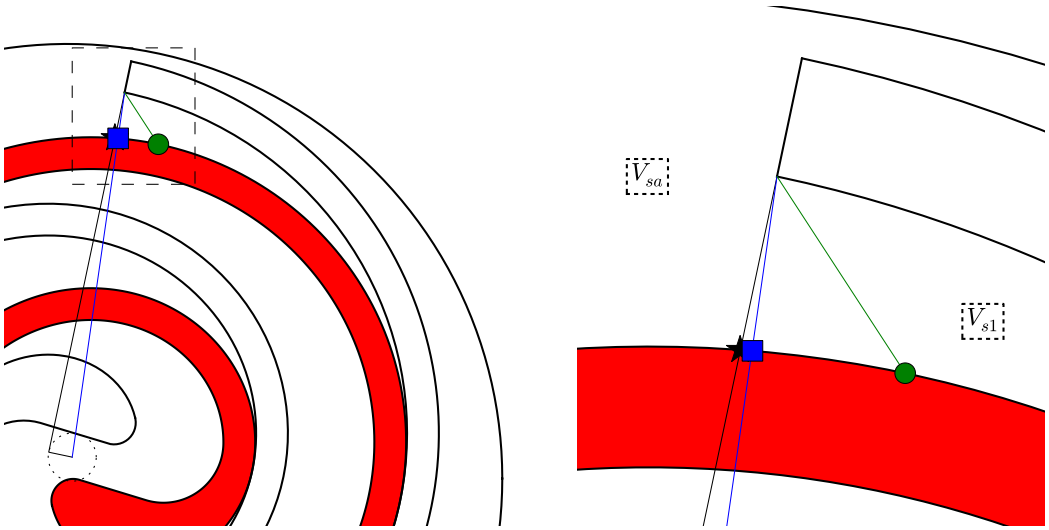
$$\begin{aligned} c_x &= \frac{1}{6A}\sum_{i=0}^{n-1}\left(x_i + x_{i+1}\right)\left(x_i y_{i+1} - x_{i+1} y_i\right) \\ c_y &= \frac{1}{6A}\sum_{i=0}^{n-1}\left(y_i + y_{i+1}\right)\left(x_i y_{i+1} - x_{i+1} y_i\right). \end{aligned} \tag{4.45}$$

4.6 Suction Chamber

Once the geometric parameters of the scroll wraps have been determined, the volumes of the compressor chambers can be determined as a function of the crank angle θ, as well as the derivatives of the volumes of the chambers with respect to the crank angle. Furthermore, the centroids of the chambers can be obtained for further use in the analysis of the dynamics and mechanical losses of the compressor.

4.6.1 Definition Of Suction Chamber Volume

Before calculating any volumes or centroids, it is necessary to discuss how the volumes of the compressor chambers are defined. In particular, this is a challenge for the suction chamber, for which there are a number of plausible definitions, some of which are commonly used but are significantly in error. The definition of the suction chamber employed has a large impact on the complexity of the mathematics involved. Figure 4.14 shows three different definitions for the line which separates the suction chamber V_{s1} from the suction area V_{sa}. There is no exact mathematical definition for the limits of the suction chamber volume while the suction chamber is open to the suction area. The three possible definitions are:



(a) Overview ($\theta = \pi/2$) (b) Inset from Overview

Figure 4.14. Definitions of suction chamber volumes.

- The break involute angle is defined to be equal to $\phi_{s,sa} = \phi_{ie} - \pi$. This yields a simple mathematical form. This point is noted with symbol ◯ in Figure 4.14(b). This is the definition used by Blunier (2006; 2009), but it yields an unphysical suction chamber volume evolution, though it does yield the correct compressor displacement.

- The break involute angle is defined based on the intersection of the tangent line from the base circle corresponding to an involute angle of the inner involute ending angle. This point is noted with symbol $\star$ in Figure 4.14(b). The derivation of this term can be found in Appendix B.2. The approximate value of $\phi_{s,sa}$ corresponding to this definition is

$$\phi_{s,sa} = \phi_e - \pi + \frac{r_o/r_b}{\phi_{ie} - \pi - \phi_{o0}} \sin\theta \tag{4.46}$$

- The break involute angle is defined based on the intersection of a line from the origin of the coordinate system to the end of the inner involute of the fixed scroll. This point is noted with symbol $\square$ in Figure 4.14(b). The derivation of this term can be found in Appendix B.2. The value of $\phi_{s,sa}$ corresponding to this definition is

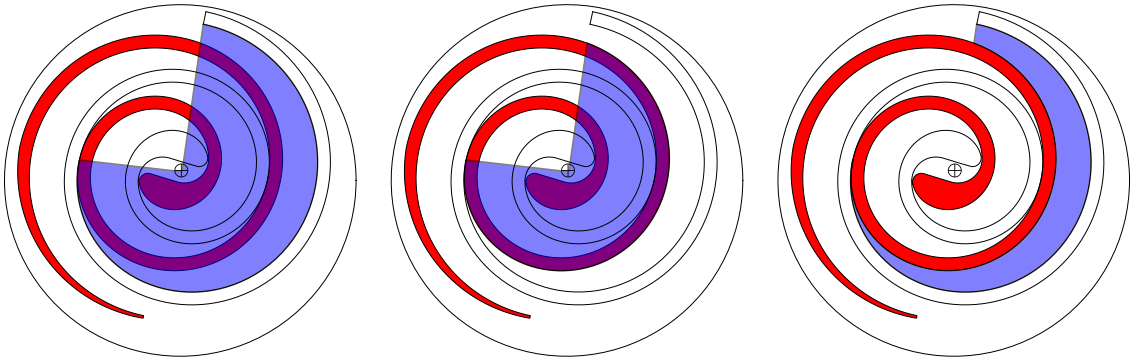
$$\begin{aligned} \phi_{s,sa} &= \phi_e - \pi + B & (4.47)\\ B &= \frac{1}{2}\left((\phi_{o0} - \phi_{ie} + \pi) + \sqrt{(\phi_{o0} - \phi_{ie} + \pi)^2 - 4D}\right) & (4.48)\\ D &= \frac{r_o}{r_b}\frac{(\phi_{i0} - \phi_{ie})\sin\theta - \cos\theta + 1}{\phi_{ie} - \phi_{i0}} & (4.49) \end{aligned}$$

  This definition yields an accurate suction chamber volume, and is the form used in the analysis which follows.

4.6.2 Suction Chamber Geometric Calculations

The suction chamber is composed of three curves - the outer involute of one wrap, the inner involute of the other wrap, and the line which closes off the suction chamber. Therefore, to calculate the volume of the suction chamber, it is possible to find the area between the origin and the outer surface of the chamber and subtract from that the area between the origin and the inner surface of the chamber. The difference is therefore the cross-sectional area of the suction chamber. Since the suction chamber has uniform cross-section, the volume is simply obtained by multiplying the area by the height of the chamber. Figure 4.15 shows the areas that are used for the s_1 chamber.

While the cross-sectional area (and the derivative of the cross-sectional area with respect to the crank angle) of both s_1 and s_2 are equivalent, s_1 will solved here. The centroids of s_1 and s_2 are not the same but symmetry can be exploited to simplify the analysis, as will be shown below.



(a) To outer surface (V_O) (b) To inner surface (V_I) (c) Remainder forms the suction chamber

Figure 4.15. Definitions of suction chamber volumes.

Outer Surface

The differential area between the origin and a parametrized curve in Cartesian coordinates with coordinates $(x(\phi), y(\phi))$ is (Kreyszig, 2006)

$$dA = \frac{1}{2}\left(x\frac{dy}{d\phi} - y\frac{dx}{d\phi}\right). \tag{4.50}$$

Therefore, for the s_1 chamber with the fixed scroll as its outer surface, the coordinates of the curve for the inner involute of the fixed scroll (x_{fi}, y_{fi}) are obtained from Eqn. (4.13), and differentiation and substitution into Eqn. (4.50) yields

$$dA_{O,s1} = \frac{r_b^2(\phi - \phi_0)^2}{2}. \tag{4.51}$$

The area is therefore given by

$$A_{O,s1} = \left.\frac{r_b^2(\phi - \phi_0)^3}{6}\right|_{\phi_1}^{\phi_2}. \tag{4.52}$$

and the involute angles which define the outer surface of s_1 to be integrated in the counter-clockwise orientation are ϕ_2=ϕ_{ie} and ϕ_1=$\phi_{ie} - \theta$, which yields the volume of

$$V_{O,s1} = \left. \frac{h_s r_b^2 (\phi - \phi_{i0})^3}{6} \right|_{\phi_{ie}-\theta}^{\phi_{ie}} \tag{4.53}$$

where h_s is the height of the scroll wrap. Substituting through yields

$$V_{O,s1} = \frac{h_s r_b^2 \left[(\phi_{ie} - \phi_{i0})^3 - (\phi_{ie} - \theta - \phi_{i0})^3 \right]}{6} \tag{4.54}$$

and the derivative of $V_{O,s1}$ with respect to the crank angle θ is

$$\frac{dV_{O,s1}}{d\theta} = \frac{h_s r_b^2}{2} (\phi_{ie} - \theta - \phi_{i0})^2. \tag{4.55}$$

The centroid of $V_{O,s1}$ is also needed to calculate the centroid of the suction chamber, which in turn is needed to be able to calculate overturning moments from the gas force on the orbiting scroll. The coordinates (c_x, c_y) of the centroid of $V_{O,s1}$ are defined by

$$c_x = \frac{\int_{\phi_1}^{\phi_2} \frac{2}{3} x dA}{\int_{\phi_1}^{\phi_2} dA} \tag{4.56}$$

$$c_y = \frac{\int_{\phi_1}^{\phi_2} \frac{2}{3} y dA}{\int_{\phi_1}^{\phi_2} dA}. \tag{4.57}$$

The factor of 2/3 enters due to the fact that the centroid of an infinitely slender arc of a circle is equal to two thirds the radius of the arc, and the integration is carried out by integrating an infinite number of differential elements which are all arcs of a circle (at least locally). The product of the coordinates of a point along an involute of the fixed scroll and the differential area between the origin and a differential element of the scroll wrap is therefore given by

$$\frac{2}{3} x dA = \frac{r_b^3}{3} (\phi - \phi_0)^2 (\cos(\phi) + (\phi - \phi_0) \sin\phi) \tag{4.58}$$

$$\frac{2}{3} y dA = \frac{r_b^3}{3} (\phi - \phi_0)^2 (\sin(\phi) - (\phi - \phi_0) \cos\phi). \tag{4.59}$$

Taking the indefinite integrals of Eqn. (4.58) and Eqn. (4.59) and dropping the constant terms for simplicity yields

$$f_{xA}(\phi, \phi_0) = \frac{r_b^3}{3} \left[4 \left((\phi - \phi_0)^2 - 2 \right) \sin\phi + (\phi_0 - \phi) \left((\phi - \phi_0)^2 - 8 \right) \cos\phi \right] \tag{4.60}$$

$$f_{yA}(\phi,\phi_0)=\frac{r_b^3}{3}\left[(\phi_0-\phi)\left((\phi-\phi_0)^2-8\right)\sin\phi-4\left((\phi-\phi_0)^2-2\right)\cos\phi\right] \quad (4.61)$$

where the results are left in functional form, into which substitutions will be made for various constituent parts. This is a general relation for any area bounded by a section of involute of the fixed scroll. For $V_{O,s1}$, the centroid can therefore be obtained from

$$c_{x,O,s1}=\frac{h_s}{V_{O,s1}}\left[f_{xA}(\phi_{ie},\phi_{i0})-f_{xA}(\phi_{ie}-\theta,\phi_{i0})\right] \quad (4.62)$$

$$c_{y,O,s1}=\frac{h_s}{V_{O,s1}}\left[f_{yA}(\phi_{ie},\phi_{i0})-f_{yA}(\phi_{ie}-\theta,\phi_{i0})\right]. \quad (4.63)$$

Inner Surface

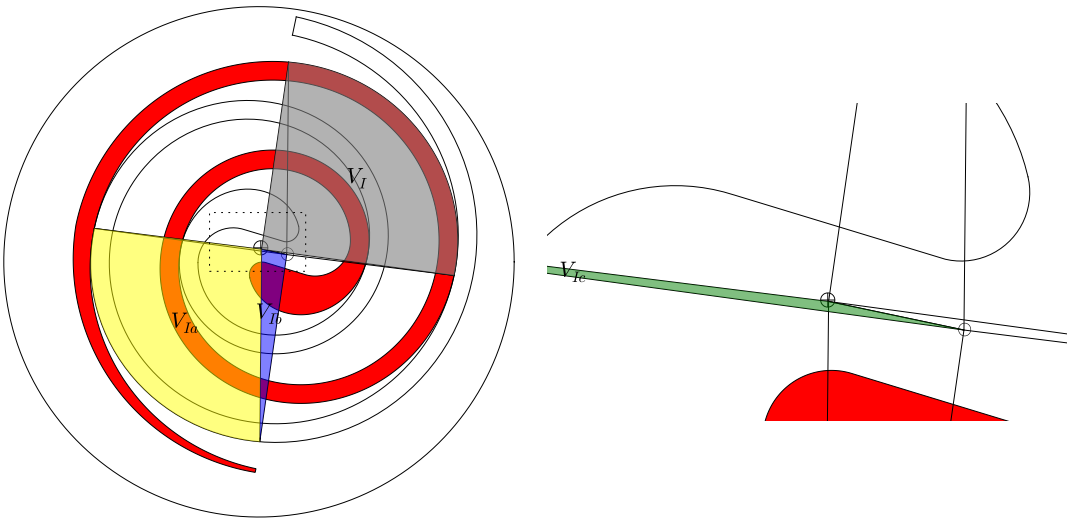
The inner surface of the suction chamber is formed by the outer involute of the orbiting scroll, where the involute angle along the outer involute of the orbiting scroll takes on the values $\phi_{ie}-\theta$ to $\phi_{s,sa}$. In order to calculate the volume $V_{I,s1}$ and $dV_{I,s1}/d\theta$, the polar integration around the origin could be taken. To simplify the analysis of the centroids for $V_{I,s1}$, a symmetric problem is solved where $V_{I,s1}$ is decomposed into three different areas $V_{Ia,s1}$, $V_{Ib,s1}$ and $V_{Ic,s1}$, as seen in Figure 4.16, where the total volume of $V_{I,s1}$ is

$$V_{I,s1}=V_{Ia,s1}+V_{Ib,s1}-V_{Ic,s1} \quad (4.64)$$

with the derivative with respect to the crank angle of

$$\frac{dV_{I,s1}}{d\theta}=\frac{dV_{Ia,s1}}{d\theta}+\frac{dV_{Ib,s1}}{d\theta}-\frac{dV_{Ic,s1}}{d\theta}. \quad (4.65)$$

This method is possible because the area between the outer involute of the orbiting scroll and the origin is the same as the area between the center of the coordinate system for the orbiting scroll and the outer involute of the fixed scroll. The advantage of this method is that $V_{Ia,s1}$ can be simply obtained from a polar integration from the origin to the outer surface of the fixed scroll. As a result, the analysis for $V_{O,s1}$ can be used for $V_{Ia,s1}$ since both are integrations between the origin and an involute of the fixed scroll. In addition, the centroids and areas of $V_{Ib,s1}$ and $V_{Ic,s1}$ can be obtained from simple analytical forms because their cross-sections are triangles with one vertex at the origin.



(a) Decomposed V_{Is1} showing volumes $V_{Ia,s1}$ and $V_{Ib,s1}$

(b) Inset from decomposition showing volume $V_{Ic,s1}$

Figure 4.16. Definitions of suction chamber volumes (volume label placed at centroid of volume).

$V_{Ia,s1}$ is simply the polar integration of an involute (in this case the outer involute of the fixed scroll) around the origin, which results in the beginning and ending involute angles of $\phi_1 = \phi_{ie} - \pi - \theta$ and $\phi_2 = \phi_{ie} - \pi + B$, respectively. From application of the same analysis as $V_{O,s1}$, and the use of Eqn. (4.53) with initial angle of ϕ_{o0} rather than ϕ_{i0}, the volume of $V_{Ia,s1}$ is given by

$$V_{Ia,s1} = \frac{h_s r_b^2}{6}\left[(\phi_{ie} - \pi + B - \phi_{o0})^3 - (\phi_{ie} - \pi - \theta - \phi_{o0})^3\right] \tag{4.66}$$

where the derivative of the volume V_{Ia} is given by

$$\frac{dV_{Ia,s1}}{d\theta} = \frac{h_s r_b^2}{2}\left[(\phi_{ie} - \pi + B - \phi_{o0})^2 B' + (\phi_{ie} - \pi - \theta - \phi_{o0})^2\right] \tag{4.67}$$

where B' is equal to the derivative of B with respect to the crank angle, or $\frac{dB}{d\theta}$, which can be obtained from Appendix B.2. The involute of $V_{Ia,s1}$ can be obtained by the same method employed for $V_{O,s1}$, which yields the centroid of

$$c_{x,Ia,s1} = \frac{h_s}{V_{Ia,s1}}\left[f_{xA}(\phi_{ie} - \pi + B, \phi_{o0}) - f_{xA}(\phi_{ie} - \pi - \theta, \phi_{o0})\right] \tag{4.68}$$

$$c_{y,Ia,s1} = \frac{h_s}{V_{Ia,s1}} \left[f_{yA}(\phi_{ie} - \pi + B, \phi_{o0}) - f_{yA}(\phi_{ie} - \pi - \theta, \phi_{o0}) \right] \tag{4.69}$$

where f_{xA} and f_{yA} are defined by Eqns. (4.60) and (4.61) respectively. Volumes $V_{Ib,s1}$ and $V_{Ic,s1}$ require simpler analysis since they are triangles with one vertex at the origin. The vertices of the triangle which forms the cross-section of $V_{Ib,s1}$ are given by (0, 0), (x_{s-sa}, y_{s-sa}), and ($r_o \cos(\phi_{ie} - \pi/2 - \theta)$, $r_o \sin(\phi_{ie} - \pi/2 - \theta)$), where x_{s-sa} and y_{s-sa} are given by

$$x_{s-sa} = r_b \left(\cos(\phi_{ie} - \pi + B) + (\phi_{ie} - \pi + B - \phi_{o0}) \sin(\phi_{ie} - \pi + B) \right) \tag{4.70}$$

$$y_{s-sa} = r_b \left(\sin(\phi_{ie} - \pi + B) - (\phi_{ie} - \pi + B - \phi_{o0}) \cos(\phi_{ie} - \pi + B) \right). \tag{4.71}$$

The area of this triangle is one-half the vector cross product of vectors going from the origin to the other two vertices of the triangle, with the order of the vectors in the cross product taken to yield a positive area from the right hand rule. This yields the volume $V_{Ib,s1}$ of

$$V_{Ib,s1} = \frac{h_s r_b r_o}{2} \left[(\phi_{ie} - \pi + B - \phi_{o0}) \sin(B + \theta) + \cos(B + \theta) \right] \tag{4.72}$$

with the derivative of volume $V_{Ib,s1}$ with respect to the crank angle of

$$\frac{dV_{Ib,s1}}{d\theta} = \frac{h_s r_b r_o (B' + 1)}{2} \left[(\phi_{ie} - \pi + B - \phi_{o0}) \cos(B + \theta) - \sin(B + \theta) \right]. \tag{4.73}$$

The centroid of a triangle is given by the average of the Cartesian coordinates of the vertices, which yields after some simplification

$$c_{x,Ib,s1} = \frac{-r_b (B - \phi_{o0} + \phi_e - \pi) \sin(B + \phi_e) - r_b \cos(B + \phi_e) - r_o \sin(\theta - \phi_e)}{3} \tag{4.74}$$

$$c_{y,Ib,s1} = \frac{-r_b \sin(B + \phi_e) + r_b (B - \phi_{o0} + \phi_e - \pi) \cos(B + \phi_e) - r_o \cos(\theta - \phi_e)}{3}. \tag{4.75}$$

The volume $V_{Ic,s1}$ is obtained by the same method as that of $V_{Ib,s1}$. Both are triangles with one vertex at the origin. The coordinates of the vertices which form the cross-section of $V_{Ic,s1}$ are (0, 0), (x_k, y_k) and ($r_o \cos(\phi_{ie} - \pi/2 - \theta)$, $r_o \sin(\phi_{ie} - \pi/2 - \theta)$) where the coordinates (x_k, y_k) are defined by

$$x_k = r_b \left(\cos(\phi_{ie} - \pi - \theta) + (\phi_{ie} - \pi - \theta - \phi_{o0}) \sin(\phi_{ie} - \pi - \theta) \right) \tag{4.76}$$

$$y_k = r_b \left(\sin(\phi_{ie} - \pi - \theta) - (\phi_{ie} - \pi - \theta - \phi_{o0})\cos(\phi_{ie} - \pi - \theta)\right). \tag{4.77}$$

Taking one-half the cross-product of the two vectors from the origin to the vertices and multiplying by the scroll wrap height yields the solution

$$V_{Ic,s1} = \frac{h_s r_b r_o}{2} \tag{4.78}$$

where the derivative of the volume $V_{Ic,s1}$ with respect to the crank angle is

$$\frac{dV_{Ic,s1}}{d\theta} = 0. \tag{4.79}$$

The centroid of $V_{Ic,s1}$ is again the mean of the coordinates of the vertices of the triangle which forms the cross-section, or, after some trigonometric simplification

$$c_{x,Ic,s1} = \frac{r_b\ (-\theta - \phi_{o0} + \phi_e - \pi)\ \sin(\theta - \phi_e) - r_o \sin(\theta - \phi_e) - r_b \cos(\theta - \phi_e)}{3} \tag{4.80}$$

$$c_{y,Ic,s1} = \frac{r_b \sin(\theta - \phi_e) + r_b\ (-\theta - \phi_{o0} + \phi_e - \pi)\ \cos(\theta - \phi_e) - r_o \cos(\theta - \phi_e)}{3}. \tag{4.81}$$

Thus, all of the constituent volumes of $V_{I,s1}$ have been calculated, which allows for the volumes $V_{Ia,s1}$, $V_{Ib,s1}$, $V_{Ic,s1}$ to be combined back into the volume $V_{I,s1}$. The volume of $V_{I,s1}$ can be obtained from Eqn. (4.64), and the centroid of V_I can be given by

$$c_{x,I,s1} = -\frac{c_{x,Ia,s1}V_{Ia,s1} + c_{x,Ib,s1}V_{Ib,s1} - c_{x,Ic,s1}V_{Ic,s1}}{V_{Ia,s1} + V_{Ib,s1} - V_{Ic,s1}} + r_o \cos(\phi_{ie} - \pi/2 - \theta) \tag{4.82}$$

$$c_{y,I,s1} = -\frac{c_{y,Ia,s1}V_{Ia,s1} + c_{y,Ib,s1}V_{Ib,s1} - c_{y,Ic,s1}V_{Ic,s1}}{V_{Ia,s1} + V_{Ib,s1} - V_{Ic,s1}} + r_o \sin(\phi_{ie} - \pi/2 - \theta) \tag{4.83}$$

where the coordinates of the centroid of $V_{I,s1}$ have been mirrored through the point of symmetry to return them to the original coordinate system.

Overall Calculations

Combining the constituent volumes which form the suction chamber s_1 yields

$$V_{s1} = V_{O,s1} - V_{I,s1} \tag{4.84}$$

and the derivative of this volume with respect to the crank angle is given by

$$\frac{dV_{s1}}{d\theta} = \frac{dV_{O,s1}}{d\theta} - \frac{dV_{I,s1}}{d\theta}. \tag{4.85}$$

Combining the centroids of the inner and outer segments therefore yields

$$c_{x,s1} = \frac{c_{x,O,s1}V_{O,s1} - c_{x,I,s1}V_{I,s1}}{V_{s1}} \tag{4.86}$$

$$c_{y,s1} = \frac{c_{y,O,s1}V_{O,s1} - c_{y,I,s1}V_{I,s1}}{V_{s1}}. \tag{4.87}$$

The symmetric suction pocket s_2 has the same volume and derivative of volume with respect to crank angle as the pocket s_1, which is expressed mathematically as

$$V_{s2} = V_{s1} \tag{4.88}$$

$$\frac{dV_{s2}}{d\theta} = \frac{dV_{s1}}{d\theta} \tag{4.89}$$

while the centroid of s_2 is the centroid of s_1 mirrored through the point of symmetry of the scroll set $\left(\frac{r_o}{2}\cos(\phi_{ie} - \pi/2 - \theta),\ \frac{r_o}{2}\sin(\phi_{ie} - \pi/2 - \theta)\right)$ yielding

$$c_{x,s2} = -c_{x,s1} + r_o\cos(\phi_{ie} - \pi/2 - \theta) \tag{4.90}$$

$$c_{y,s2} = -c_{y,s1} + r_o\sin(\phi_{ie} - \pi/2 - \theta). \tag{4.91}$$

Finally, the displacement of the compressor is defined as the sum of the volumes of both s_1 and s_2 chambers at the end of the suction process at a crank angle of $\theta = 2\pi$, which is given by

$$V_{disp} = -2\pi h_s r_b r_o \left(3\pi - 2\phi_{ie} + \phi_{i0} + \phi_{o0}\right). \tag{4.92}$$

Forces

While the compressor is under operation, the pressure of the gas in the suction chamber applies a distributed load on all the surfaces of the chamber. The instantaneous net axial force (in the z-direction) from the contributions of both s_1 and s_2 control volumes are given simply by

$$\frac{\mathbf{F}_{z,s1}}{p_{s1} - p_{shell}} = \frac{V_{s1}}{h_s} \tag{4.93}$$

$$\frac{\mathbf{F}_{z,s2}}{p_{s2} - p_{shell}} = \frac{V_{s2}}{h_s} \tag{4.94}$$

where p_{s1} and p_{s2} are the pressures of the s_1 and s_2 chambers respectively and p_{shell} is the back pressure. The x and y components of the gas force on the orbiting scroll are obtained by integrating the expressions in Eqn. (4.38), for both the s_1 and s_2 chambers along the wraps of the orbiting scroll. This yields the equations

$$\frac{\mathbf{F}_{s1}}{p_{s1}} = h_s r_b \int_{\phi_e-\pi-\theta}^{\phi_e-\pi+B} \Big[(\phi - \phi_{o0})\sin(\phi)\hat{i} - (\phi - \phi_{o0})\cos(\phi)\hat{j}\Big]\, d\phi \tag{4.95}$$

$$\frac{\mathbf{F}_{s2}}{p_{s2}} = -h_s r_b \int_{\phi_e-\theta}^{\phi_e} \Big[(\phi - \phi_{i0})\sin(\phi)\hat{i} + (\phi - \phi_{i0})\cos(\phi)\hat{j}\Big]\, d\phi \tag{4.96}$$

which when integrated yield the net forces for the suction chambers given in Appendix B.4. The coordinates of the pin on the crank are given by Eqns. (4.40) and (4.41) and the torque generated by the suction chambers can be obtained from

$$\frac{\mathbf{M}_{\bigcirc,s1}}{p_{s1}} = \int_{\phi_e-\pi-\theta}^{\phi_e-\pi+B} [\mathbf{r}_{\bigcirc,s1} \times \frac{d\mathbf{F}_{s1}}{p_{s1}}] d\phi \tag{4.97}$$

$$\frac{\mathbf{M}_{\bigcirc,s2}}{p_{s2}} = \int_{\phi_e-\theta}^{\phi_e} [\mathbf{r}_{\bigcirc,s2} \times \frac{d\mathbf{F}_{s2}}{p_{s2}}] d\phi \tag{4.98}$$

which yields the moments

$$\frac{\mathbf{M}_{\bigcirc,s1}}{p_{s1}} = \frac{r_b^2\, h_s\, (B - \theta - 2\,\phi_{o0} + 2\,\phi_e - 2\,\pi)\, (B + \theta)}{2}\hat{k} \tag{4.99}$$

$$\frac{\mathbf{M}_{\bigcirc,s2}}{p_{s2}} = \frac{r_b^2\, h_s\, \theta\, (\theta + 2\,\phi_{i0} - 2\,\phi_e)}{2}\hat{k}. \tag{4.100}$$

4.7 Suction Area Chamber

The suction area chamber *sa* does not play a large role in the working processes of the scroll compressor. This chamber is necessary only to allow flow to get from the single inlet port to the two parallel suction pockets. In addition, it is generally partially filled with other metal from the casting process, so it does not take the full volume. The volume contained by the scroll sets can be approximated as being twice that shown in Figure 4.17 with the bounding involute angles of ϕ_{ie} and ϕ_{s-sa}. A slight error is introduced because the $s - sa$ break angles are not the same for fixed and orbiting scroll outer involutes. In addition, there is a sliver between the origin

and the inner and outer ending angles of the fixed scroll. With these approximations, the shaded volume can be given by

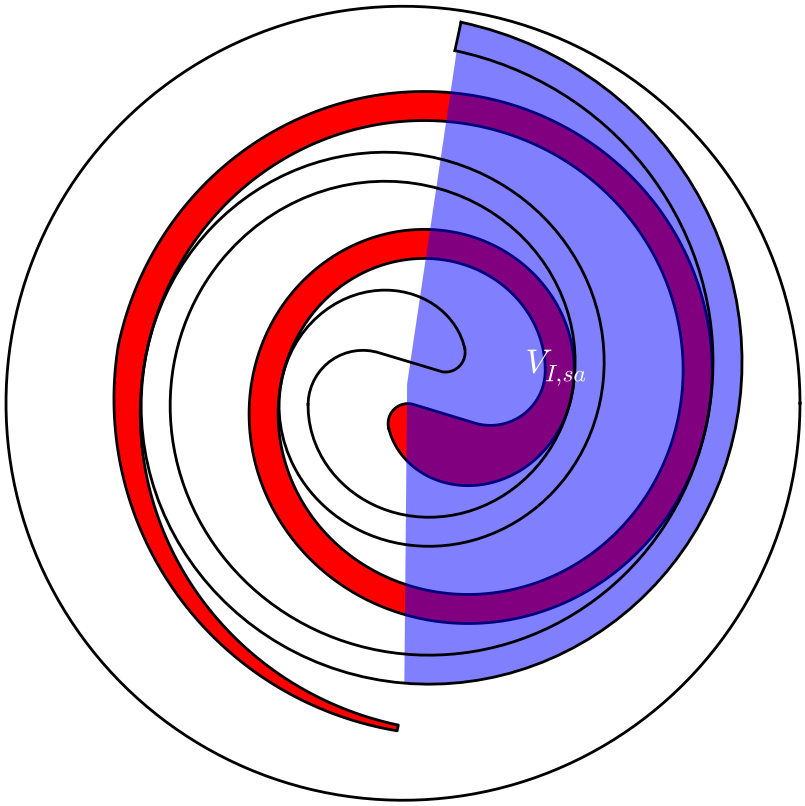


Figure 4.17. Definition of inner volume of V_{sa}.

$$V_{I,sa} = \frac{h_s r_b^2}{6}\left[(\phi_{ie} - \phi_{o0})^3 - (\phi_{ie} - \pi + B - \phi_{o0})^3\right] \tag{4.101}$$

since it is an area integration around the origin to a fixed scroll involute. The derivative of the volume V_{Ia} is given by

$$\frac{dV_{I,sa}}{d\theta} = -\frac{h_s r_b^2}{2}(\phi_{ie} - \pi + B - \phi_{o0})^2 B' \tag{4.102}$$

where B' can be obtained from Appendix B.2 and the total volume and derivative of the volume of the *sa* chamber are equal to

$$V_{sa} = \pi h_s \frac{D_{wall}^2}{4} - 2V_{I,sa} \tag{4.103}$$

$$\frac{dV_{sa}}{d\theta} = -2\frac{dV_{I,sa}}{d\theta} \tag{4.104}$$

where D_{wall} is the inner diameter of the shell into which the scroll set is placed.

Forces

The forces on the orbiting scroll from the *sa* chamber are given by

$$\frac{\mathbf{F}_{sa}}{p_{sa}} = h_s r_b \int_{\phi_e-\pi+B}^{\phi_e} \left[(\phi - \phi_{o0})\sin(\phi)\hat{i} - (\phi - \phi_{o0})\cos(\phi)\hat{j}\right] d\phi \tag{4.105}$$

which when integrated yield the net forces for the suction chambers given in Appendix B.4. The moment around the crank pin generated by the portion of the involute for the *sa* chamber is equal to

$$\frac{\mathbf{M}_{\bigcirc,sa}}{p_{sa}} = \int_{\phi_e-\pi+B}^{\phi_e} [\mathbf{r}_{\bigcirc,sa} \times \frac{d\mathbf{F}_{sa}}{p_{sa}}] d\phi \tag{4.106}$$

where the moments are taken around the point from Eqn. (4.40) which yields

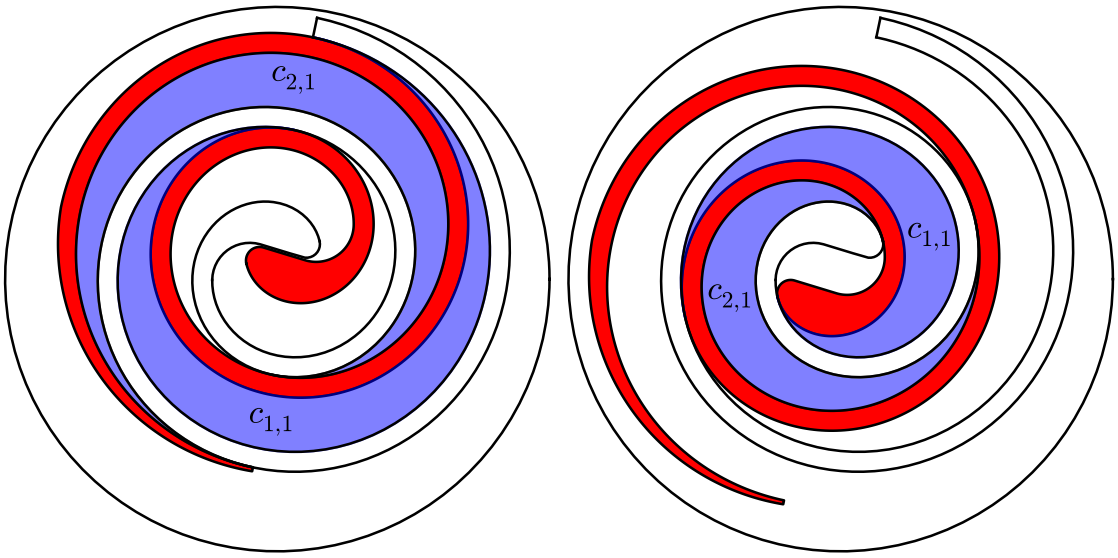
$$\frac{\mathbf{M}_{\bigcirc,sa}}{p_{sa}} = -\frac{r_b^2\, h_s\, (B-\pi)\, (B - 2\,\phi_{o0} + 2\,\phi_e - \pi)}{2}\hat{k}. \tag{4.107}$$

4.8 Compression Chambers

For the compression chambers, as long as they exist, the volume is a sealed off pocket with cross-section areas resembling that of Figure 4.18 with involutes forming the outer and inner surfaces. The number of pairs of compression chambers in existence at a given crank angle θ is given by

$$N_c = floor\left(\frac{\phi_{ie} - \theta - \phi_{os} - \pi}{2\pi}\right) \tag{4.108}$$

and is found by solving Eqn. (4.15) for the number of pairs of compression chambers in existence at the crank angle θ. In Figure 4.18, there is one pair of compression chambers in existence. For each compression chamber, the involute angles for the involute which forms the outer involute of the compression chamber of the integration are $\phi_{\alpha+1}$ to ϕ_α where for the outermost chamber, the value of α is 1. The value of the conjugate angles (with $k = \alpha$) can be obtained from Eqn. (4.6). The second compression chamber towards the center of the compressor has involute angle limits



(a) θ=0 (b) $\theta = \theta_d$

Figure 4.18. Compression Chambers.

of $\phi_{\alpha+2}$ to $\phi_{\alpha+1}$, and so on until the last conjugate angle before the starting angle is reached.

The approach for solving the compression chambers is slightly different than that applied to the suction chambers as a consequence of the more straight-forward geometry and bounding involute angles. With the suction chambers, the analysis is carried out for the c_1 chamber bounded on its outside by the inner involute of the fixed scroll, and symmetry is then used to solve for c_2. The mathematics for the centroids of the compression chambers are sufficiently straightforward for the compression chambers such that inner volume decomposition is not required as is the case for the suction chambers.

The differential area of an arc enclosed by a point on the inner involute of the fixed scroll is obtained from

$$dA_O = \frac{r_b^2(\phi - \phi_{i0})^2}{2} d\phi \tag{4.109}$$

and the differential area of an arc with center at the origin and enclosed by the outer involute of the orbiting scroll is

$$dA_I = \frac{r_b^2(\phi - \phi_{o0})^2 - r_b r_o(\phi - \phi_{o0})\cos(\theta - \phi_{ie} + \phi)}{2} d\phi. \tag{4.110}$$

For the α-th compression chamber, the volume is therefore given by

$$V_{c1,\alpha} = h_s \left[\int_{\phi_{ie}-\theta-2\pi\alpha}^{\phi_{ie}-\theta-(\alpha-1)2\pi} \frac{dA_O}{d\phi} d\phi - \int_{\phi_{ie}-\theta-2\pi\alpha-\pi}^{\phi_{ie}-\theta-(\alpha-1)2\pi-\pi} \frac{dA_I}{d\phi} d\phi \right]. \tag{4.111}$$

After integration and simplification, the volume of the α-th compression chamber is determined to be

$$V_{c1,\alpha} = -\pi h_s r_b r_o \left(2\theta + 4\pi\alpha - 2\phi_{ie} - \pi + \phi_{i0} + \phi_{o0}\right) \tag{4.112}$$

and the derivative of the volume of the compression chamber is given as

$$\frac{dV_{c1,\alpha}}{d\theta} = -2\pi h_s r_b r_o \tag{4.113}$$

which has no dependence on the crank angle or compression chamber index α, so the volume of all the compression chambers decrease linearly with the crank angle θ. In order to calculate the built-in volume ratio of the scroll compressor, the discharge angle from Eqn. (4.17) is substituted into Eqn. (4.112) with $\alpha = N_{c,max}$, and the volume of the innermost compression chamber at the discharge angle is

$$V_{c1,d} = -\pi h_s r_b r_o \left(-2\phi_{os} - 3\pi + \phi_{i0} + \phi_{o0}\right) \tag{4.114}$$

and thus, the built-in volume ratio, an important design parameter, is given by

$$V_{ratio} = \frac{V_{disp}}{2V_{c1,d}} = \frac{3\pi - 2\phi_{ie} + \phi_{i0} + \phi_{o0}}{-2\phi_{os} - 3\pi + \phi_{i0} + \phi_{o0}}. \tag{4.115}$$

In order to calculate the centroid of the α-th compression chamber, the volume is broken up into two pieces, the area enclosed by the inner involute and the origin for the fixed scroll and the area enclosed by the outer involute and the origin for the fixed

scroll. The coordinates of the centroid for the compression chamber $c_{1,\alpha}$ are therefore given by

$$c_{x,c1,\alpha} = \frac{2h_s}{3} \frac{\int_{\phi_e-\theta-(\alpha-1)2\pi}^{\phi_e-\theta-2\pi\alpha} x_{oi} \frac{dA_I}{d\phi} d\phi - \int_{\phi_e-\theta-2\pi\alpha-\pi}^{\phi_e-\theta-(\alpha-1)2\pi-\pi} x_{oo} \frac{dA_O}{d\phi} d\phi}{V_{c1,\alpha}} \tag{4.116}$$

$$c_{y,c1,\alpha} = \frac{2h_s}{3} \frac{\int_{\phi_e-\theta-(\alpha-1)2\pi}^{\phi_e-\theta-2\pi\alpha} y_{oi} \frac{dA_I}{d\phi} d\phi - \int_{\phi_e-\theta-2\pi\alpha-\pi}^{\phi_e-\theta-(\alpha-1)2\pi-\pi} y_{oo} \frac{dA_O}{d\phi} d\phi}{V_{c1,\alpha}}. \tag{4.117}$$

which yields the solution

$$c_{x,c1,\alpha} = -2r_b \cos(\theta - \phi_{ie}) - \Psi \sin(\theta - \phi_{ie}) \tag{4.118}$$

$$c_{y,c1,\alpha} = 2r_b \sin(\theta - \phi_{ie}) - \Psi \cos(\theta - \phi_{ie}) \tag{4.119}$$

where the term Ψ is defined for compactness by

$$\Psi = \frac{r_b}{3} \frac{3(\theta + \phi_{o0})^2 + \pi^2 - 15 + (\theta + \phi_{o0})(12\pi\alpha - 6\phi_{ie}) + 3\,\phi_{ie}^2 + 12\pi\alpha(\pi\alpha - \phi_{ie})}{2\theta + \phi_{o0} + \phi_{i0} - 2\phi_{ie} + 4\pi\alpha - \pi}. \tag{4.120}$$

The volume and derivative of chamber volume for the $c_{2,\alpha}$ chamber is given by

$$V_{c2,\alpha} = V_{c1,\alpha} \tag{4.121}$$

$$\frac{dV_{c2,\alpha}}{d\theta} = \frac{dV_{c1,\alpha}}{d\theta} \tag{4.122}$$

by symmetry. This yields the centroid for the $c_{2,\alpha}$ chamber of

$$c_{x,c2,\alpha} = -c_{x,c1,\alpha} + r_o \cos(\phi_{ie} - \pi/2 - \theta) \tag{4.123}$$

$$c_{y,c2,\alpha} = -c_{y,c1,\alpha} + r_o \sin(\phi_{ie} - \pi/2 - \theta). \tag{4.124}$$

Forces

The gas forces on the orbiting scroll from the compression chambers are calculated by a similar means to that for the suction chambers. The axial forces are given by

$$\frac{\mathbf{F}_{z,c1,\alpha}}{p_{c1,\alpha} - p_{shell}} = \frac{V_{c1,\alpha}}{h_s} \tag{4.125}$$

$$\frac{\mathbf{F}_{z,c2,\alpha}}{p_{c2,\alpha} - p_{shell}} = \frac{V_{c2,\alpha}}{h_s} \tag{4.126}$$

and the in-plane forces given by

$$\frac{\mathbf{F}_{c1,\alpha}}{p_{c1,\alpha}} = h_s r_b \int_{\phi_e-\theta-2\pi\alpha-\pi}^{\phi_e-\theta-2\pi(\alpha-1)-\pi} \left[(\phi-\phi_{o0})\sin(\phi)\hat{i} - (\phi-\phi_{o0})\cos(\phi)\hat{j}\right] d\phi \tag{4.127}$$

$$\frac{\mathbf{F}_{c2,\alpha}}{p_{c2,\alpha}} = -h_s r_b \int_{\phi_e-\theta-2\pi\alpha}^{\phi_e-\theta-2\pi(\alpha-1)} \left[(\phi-\phi_{i0})\sin(\phi)\hat{i} + (\phi-\phi_{i0})\cos(\phi)\hat{j}\right] d\phi \tag{4.128}$$

and the solutions are found in Appendix B.4. The moments around the crank pin are given by

$$\frac{\mathbf{M}_{\bigcirc,c1,\alpha}}{p_{c1,\alpha}} = \int_{\phi_e-\theta-2\pi\alpha-\pi}^{\phi_e-\theta-2\pi(\alpha-1)-\pi} [\mathbf{r}_{\bigcirc,c1,\alpha} \times \frac{d\mathbf{F}_{c1,\alpha}}{p_{c1,\alpha}}] d\phi \tag{4.129}$$

$$\frac{\mathbf{M}_{\bigcirc,c2,\alpha}}{p_{c2,\alpha}} = \int_{\phi_e-\theta-2\pi\alpha}^{\phi_e-\theta-2\pi(\alpha-1)} [\mathbf{r}_{\bigcirc,c2,\alpha} \times \frac{d\mathbf{F}_{c2,\alpha}}{p_{c2,\alpha}}] d\phi \tag{4.130}$$

where the moments are taken around the point from Eqn. (4.40) which yields the moments around the crank pin

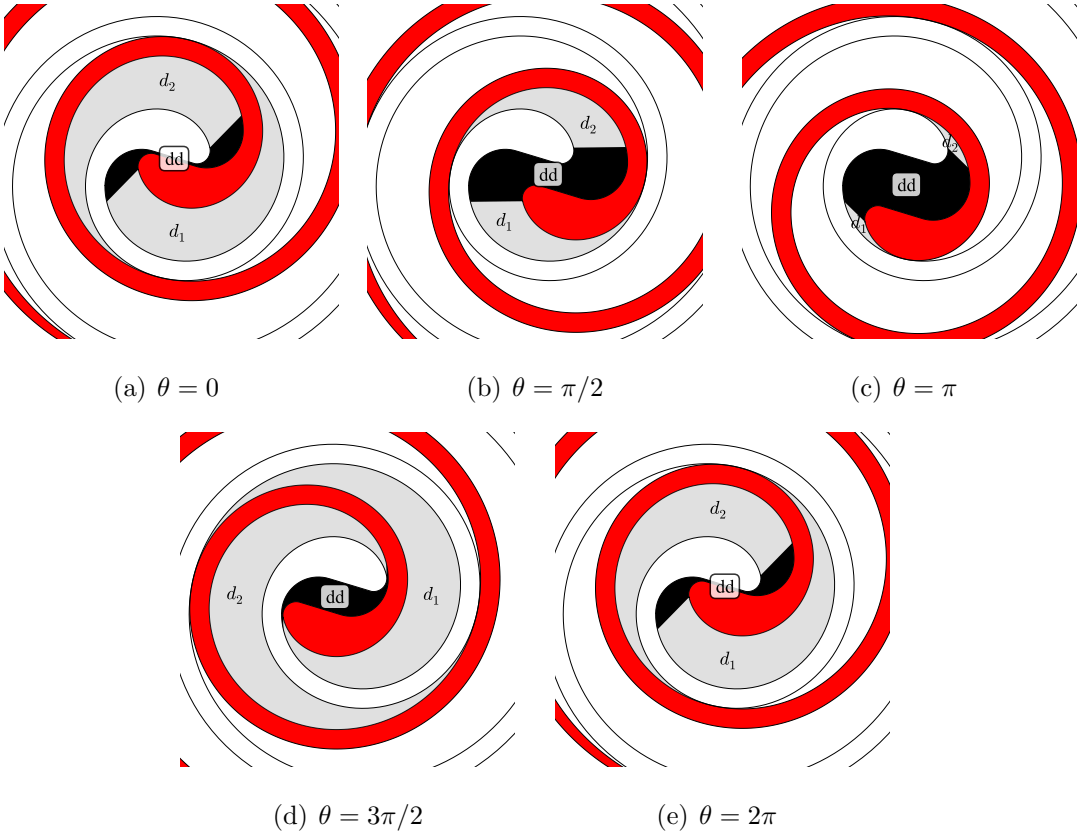
$$\frac{\mathbf{M}_{\bigcirc,c1,\alpha}}{p_{c1,\alpha}} = -2\,\pi\,r_b^2\,h_s\,\left(\theta+\phi_{o0}-\phi_e+2\,\pi\,\alpha-\pi\right)\hat{k} \tag{4.131}$$

$$\frac{\mathbf{M}_{\bigcirc,c2,\alpha}}{p_{c2,\alpha}} = \frac{\pi\,r_b^2\,h_s\,\left(2\,\theta+2\,\phi_{i0}-2\,\phi_e+4\,\pi\,\alpha+\pi\right)}{2}\hat{k}. \tag{4.132}$$

4.9 Discharge Region

For the suction and compression chambers, the geometry is governed by involute curves and the interactions of these curves define the boundaries of the scroll compressor chambers. In the discharge region, there is no longer involute-involute contact to form the inner-most boundary of the chamber, and the discharge region can be broken into three volumes, d_1 and d_2, which are two symmetric discharge pockets created from the redefinition of compression chambers at the discharge angle and the *dd* chamber which is directly connected to the discharge port. Figure 4.19 shows the definition of the discharge chambers over the course of one rotation.

At the discharge angle, a new set of d_1/d_2 chambers are created from the inner-most compression chambers, and a rapid pressure equilibration occurs between the chambers. Once the pressures of the chambers have equilibrated to within some



(a) $\theta = 0$ (b) $\theta = \pi/2$ (c) $\theta = \pi$

(d) $\theta = 3\pi/2$ (e) $\theta = 2\pi$

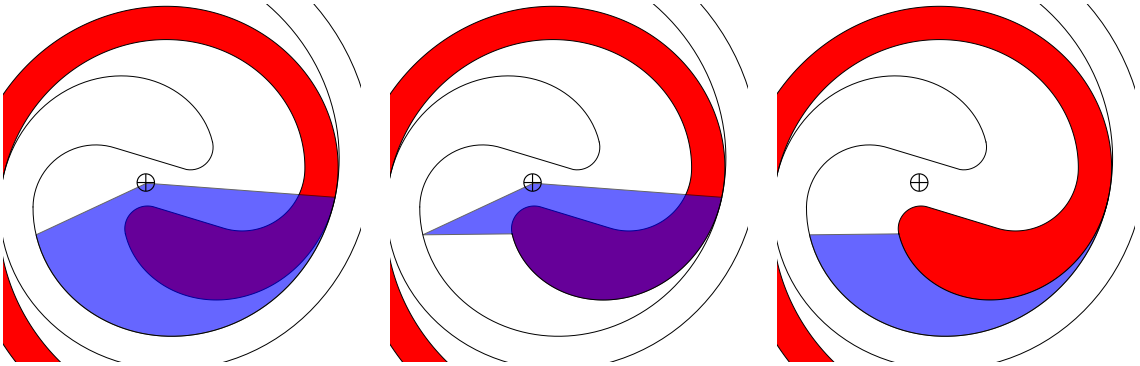
Figure 4.19. Discharge chamber definitions over one rotation.

tolerance, the volumes of the d_1, d_2 and dd chambers are merged and the discharge region is considered to have only one discharge chamber with volume equal to the sum of the volumes of the three constituent volumes.

4.10 Discharge Chambers d_1 And d_2

As with the compression and suction chambers, the analysis for volumes and centroids is carried out for the d_1 chamber, and symmetry is used to solve for the necessary parameters for d_2.

For the d_1 chamber, the dividing line between the d_1 and dd chambers is taken to be the line which connects ϕ_{os} on the orbiting scroll to $\phi_{os} + \pi$ on fixed scroll. This condition ensures that the line separating the chambers has a length of 0 at θ=θ_d.



(a) To outer surface $V_{O,d1}$ (b) To inner surface $V_{I,d1}$ (c) Difference forms the d_1 chamber

Figure 4.20. Definitions of chamber d_1 volumes.

This definition for the dividing line begins to break down at crank angles significantly greater than the discharge angle, but the pressure equilibration process typically occurs within a quarter-rotation ($\pi/2$ radians) of the crank, so using this chamber definition is acceptable as it greatly simplifies the analysis. Thus, the outer boundary of the chamber is defined by the involute angles of $\phi_1 = \phi_{os} + \pi$ to $\phi_2 = \phi_{ie} - \theta - 2\pi N_c$ where N_c is defined from Eqn. (4.108). As the volume enclosed by the outer surface of the chamber is enclosed by the origin and an involute curve of the fixed scroll, the analysis of Section 4.6.2 can be used to determine the volume of the outer portion, given by

$$V_{O,d1} = \frac{h_s r_b^2 \left[(\phi_{ie} - \theta - 2\pi N_c - \phi_{i0})^3 - (\phi_{os} + \pi - \phi_{i0})^3\right]}{6} \tag{4.133}$$

and the derivative of $V_{O,d1}$ with respect to the crank angle is

$$\frac{dV_{O,d1}}{d\theta} = -\frac{h_s r_b^2}{2}(\phi_{ie} - \theta - 2\pi N_c - \phi_{i0})^2. \tag{4.134}$$

The centroids can also be calculated by the method proposed in Section 4.6.2, which yields

$$c_{x,O,d1} = \frac{h_s}{V_{O,d1}} \left[f_{xA}(\phi_{ie} - \theta - 2\pi N_c, \phi_{i0}) - f_{xA}(\phi_{os} + \pi, \phi_{i0})\right] \tag{4.135}$$

$$c_{y,O,d1} = \frac{h_s}{V_{O,d1}} \left[f_{yA}(\phi_{ie} - \theta - 2\pi N_c, \phi_{i0}) - f_{yA}(\phi_{os} + \pi, \phi_{i0})\right]. \tag{4.136}$$

The inner portion of the chamber $V_{Ia,d1}$, is defined by involute angles of $\phi_1 = \phi_{os}$ to

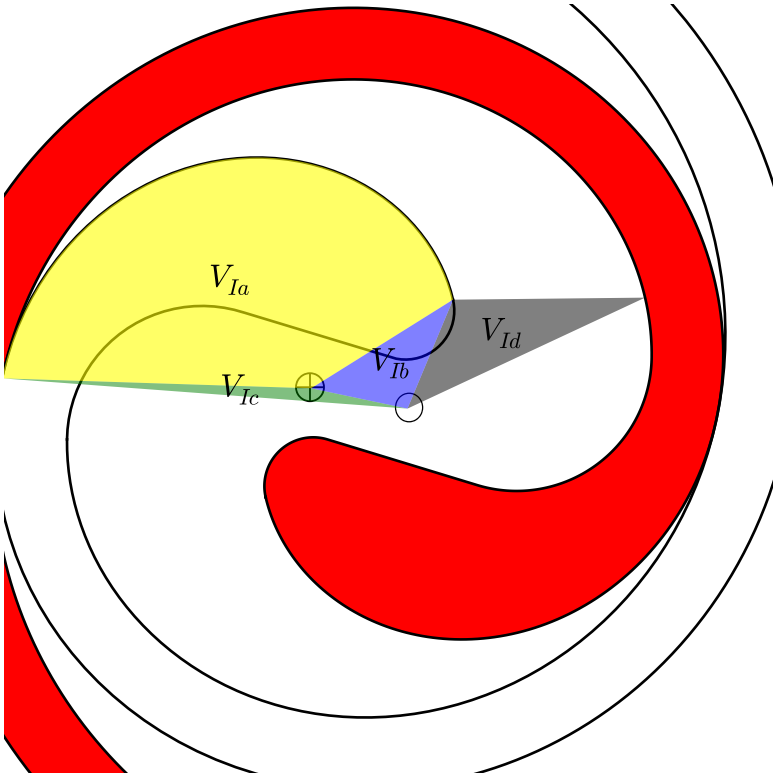


Figure 4.21. Decomposition of chamber d_1.

$\phi_2 = \phi_{ie} - \theta - 2\pi N_c - \pi$ where N_c is defined from Eqn. (4.108). Applying the same analysis as for the suction chamber yields the volume of $V_{Ia,d1}$ of

$$V_{Ia,d1} = \frac{h_s r_b^2 \left[(\phi_{ie} - \theta - 2\pi N_c - \pi - \phi_{o0})^3 - (\phi_{os} - \phi_{o0})^3\right]}{6} \tag{4.137}$$

and the derivative of $V_{Ia,d1}$ with respect to the crank angle is

$$\frac{dV_{Ia,d1}}{d\theta} = -\frac{h_s r_b^2}{2}(\phi_{ie} - \theta - 2\pi N_c - \pi - \phi_{o0})^2. \tag{4.138}$$

Applying the analysis of Section 4.6.2 for the centroid of area enclosed between an involute curve and the fixed origin yields

$$c_{x,Ia,d1} = \frac{h_s}{V_{Ia,d1}} \left[f_{xA}(\phi_{ie} - \theta - 2\pi N_c - \pi, \phi_{o0}) - f_{xA}(\phi_{os}, \phi_{o0})\right] \tag{4.139}$$

$$c_{y,Ia,d1} = \frac{h_s}{V_{Ia,d1}} \left[f_{yA}(\phi_{ie} - \theta - 2\pi N_c - \pi, \phi_{o0}) - f_{yA}(\phi_{os}, \phi_{o0})\right]. \tag{4.140}$$

The volume $V_{Ib,d1}$ is a triangle with vertices at the origin, the origin for the orbiting scroll, and the outer starting angle of the fixed scroll. As a result, using the analysis of Section 4.5.1, the necessary terms are

$$V_{Ib,d1} = \frac{h_s r_b\, r_o}{2}\left[(\phi_{os} - \phi_{o0})\sin(\theta + \phi_{os} - \phi_{ie}) + \cos(\theta + \phi_{os} - \phi_{ie})\right] \tag{4.141}$$

$$\frac{dV_{Ib,d1}}{d\theta} = \frac{h_s r_b\, r_o}{2}\left[(\phi_{os} - \phi_{o0})\cos(\theta + \phi_{os} - \phi_{ie}) - \sin(\theta + \phi_{os} - \phi_{ie})\right] \tag{4.142}$$

$$c_{x,Ib,d1} = \frac{-r_o \sin(\theta - \phi_{ie}) + r_b\ (\phi_{os} - \phi_{o0})\ \sin(\phi_{os}) + r_b \cos(\phi_{os})}{3} \tag{4.143}$$

$$c_{y,Ib,d1} = \frac{-r_o \cos(\theta - \phi_{ie}) - r_b\ (\phi_{os} - \phi_{o0})\ \cos(\phi_{os}) + r_b \sin(\phi_{os})}{3}. \tag{4.144}$$

The volume $V_{Ic,d1}$ is also formed by a triangle, in this case with vertices at the origin, the innermost conjugate point on the outer involute of the fixed scroll, and the orbiting scroll origin. Thus, the governing equations for this triangle are

$$V_{Ic,d1} = \frac{h_s r_b\, r_o}{2} \tag{4.145}$$

$$\frac{dV_{Ic,d1}}{d\theta} = 0 \tag{4.146}$$

$$c_{x,Ic,d1} = \frac{[r_b\ (-\theta + \phi_{ie} - \phi_{o0} - 2\,\pi\,N_c - \pi) - r_o]\sin(\theta - \phi_{ie}) - r_b \cos(\theta - \phi_{ie})}{3} \tag{4.147}$$

$$c_{y,Ic,d1} = \frac{[r_b\ (-\theta + \phi_{ie} - \phi_{o0} - 2\,\pi\,N_c - \pi) - r_o]\cos(\theta - \phi_{ie}) + r_b \sin(\theta - \phi_{ie})}{3}. \tag{4.148}$$

Finally, the triangle which forms the cross-section of $V_{Id,d1}$ is needed. This triangle has one vertex at the orbiting scroll origin, one at the inner starting angle of the orbiting scroll, and another at the outer starting angle of the fixed scroll. Thus, the governing equations for $V_{Id,d1}$ are

$$V_{Id,d1} = \frac{h_s r_b r_o}{2}\left[(\phi_{os} - \phi_{i0} + \pi)\sin(\theta + \phi_{os} - \phi_{ie}) + \cos(\theta + \phi_{os} - \phi_{ie}) + 1\right] \tag{4.149}$$

$$\frac{dV_{Id,d1}}{d\theta} = \frac{h_s r_b r_o}{2}\left[(\phi_{os} - \phi_{i0} + \pi)\cos(\theta + \phi_{os} - \phi_{ie}) - \sin(\theta + \phi_{os} - \phi_{ie})\right] \tag{4.150}$$

$$c_{x,Id,d1} = \frac{r_b\ (2\,\phi_{os} - \phi_{o0} - \phi_{i0} + \pi)\ \sin(\phi_{os}) - 2\ (r_o \sin(\theta - \phi_{ie}) - r_b \cos(\phi_{os}))}{3} \tag{4.151}$$

$$c_{y,Id,d1} = \frac{-2\ (r_o \cos(\theta - \phi_{ie}) - r_b \sin(\phi_{os})) - r_b\ (2\,\phi_{os} - \phi_{o0} - \phi_{i0} + \pi)\ \cos(\phi_{os})}{3}. \tag{4.152}$$

Combining the centroid components for the inner volume for the d_1 chamber yields

$$c_{x,I,d1} = -\frac{c_{x,Ia,d1}V_{Ia,d1} + c_{x,Ib,d1}V_{Ib,d1} + c_{x,Ic,d1}V_{Ic,d1} + c_{x,Id,d1}V_{Id,d1}}{V_{Ia,d1} + V_{Ib,d1} + V_{Ic,d1} + V_{Id,d1}} + r_o \cos(\theta_m) \tag{4.153}$$

$$c_{y,I,d1} = -\frac{c_{y,Ia,s1}V_{Ia,d1} + c_{y,Ib,d1}V_{Ib,d1} + c_{y,Ic,d1}V_{Ic,d1} + c_{y,Id,d1}V_{Id,d1}}{V_{Ia,d1} + V_{Ib,d1} + V_{Ic,d1} + V_{Id,d1}} + r_o \sin(\theta_m) \tag{4.154}$$

and the volume of the inner segment is equal to

$$V_{I,d1} = V_{Ia,d1} + V_{Ib,d1} + V_{Ic,d1} + V_{Id,d1} \tag{4.155}$$

$$\frac{dV_{I,d1}}{d\theta} = \frac{dV_{Ia,d1}}{d\theta} + \frac{dV_{Ib,d1}}{d\theta} + \frac{dV_{Ic,d1}}{d\theta} + \frac{dV_{Id,d1}}{d\theta}. \tag{4.156}$$

The total volume for the d_1 chamber can therefore be obtained from

$$V_{d1} = V_{O,d1} - V_{I,d1} \tag{4.157}$$

$$\frac{dV_{d1}}{d\theta} = \frac{dV_{O,d1}}{d\theta} - \frac{dV_{I,d1}}{d\theta} \tag{4.158}$$

and the centroid of the d_1 chamber is given by

$$c_{x,d1} = \frac{c_{x,O,d1}V_{O,d1} - c_{x,I,d1}V_{I,d1}}{V_{d1}} \tag{4.159}$$

$$c_{y,d1} = \frac{c_{y,O,d1}V_{O,d1} - c_{y,I,d1}V_{I,d1}}{V_{d1}} \tag{4.160}$$

By symmetry, the centroid for the d_2 chamber is

$$c_{x,d2} = -c_{x,d1} + r_o \cos(\phi_{ie} - \pi/2 - \theta) \tag{4.161}$$

$$c_{y,d2} = -c_{y,d1} + r_o \sin(\phi_{ie} - \pi/2 - \theta). \tag{4.162}$$

and the volume and derivative of volume for the d_2 chamber are given by

$$V_{d2} = V_{d1} \tag{4.163}$$

$$\frac{dV_{d2}}{d\theta} = \frac{dV_{d1}}{d\theta}. \tag{4.164}$$

Forces

The gas forces on the orbiting scroll from the discharge chambers are calculated by a similar means to that for the other chambers. The axial forces are given by

$$\frac{\mathbf{F}_{z,d1}}{p_{d1} - p_{shell}} = \frac{V_{d1}}{h_s} \tag{4.165}$$

$$\frac{\mathbf{F}_{z,d2}}{p_{d2} - p_{shell}} = \frac{V_{d2}}{h_s} \tag{4.166}$$

and the in-plane forces given by

$$\frac{\mathbf{F}_{d1}}{p_{d1}} = h_s r_b \int_{\phi_{os}}^{\phi_e - \theta - 2\pi N_c - \pi} \left[(\phi - \phi_{o0}) \sin(\phi) \hat{i} - (\phi - \phi_{o0}) \cos(\phi) \hat{j} \right] d\phi \tag{4.167}$$

$$\frac{\mathbf{F}_{d2}}{p_{d2}} = -h_s r_b \int_{\phi_{os}+\pi}^{\phi_e - \theta - 2\pi N_c} \left[(\phi - \phi_{i0}) \sin(\phi) \hat{i} + (\phi - \phi_{i0}) \cos(\phi) \hat{j} \right] d\phi \tag{4.168}$$

where solutions for the force terms are in Appendix B.4. The moments around the crank pin are given by

$$\frac{\mathbf{M}_{\bigcirc,d1}}{p_{d1}} = \int_{\phi_{os}}^{\phi_e - \theta - 2\pi N_c - \pi} [\mathbf{r}_{\bigcirc,d1} \times \frac{d\mathbf{F}_{d1}}{p_{d1}}] d\phi \tag{4.169}$$

$$\frac{\mathbf{M}_{\bigcirc,d2}}{p_{d2}} = \int_{\phi_{os}+\pi}^{\phi_e - \theta - 2\pi N_c} [\mathbf{r}_{\bigcirc,d2} \times \frac{d\mathbf{F}_{d2}}{p_{d2}}] d\phi \tag{4.170}$$

which yields the torques

$$\frac{\mathbf{M}_{\bigcirc,d1}}{p_{d1}} = \frac{r_b^2 \, h_s \, (\theta - \phi_{os} + 2\,\phi_{o0} - \phi_e + 2\,\pi\,N_c + \pi) \, (\theta + \phi_{os} - \phi_e + 2\,\pi\,N_c + \pi)}{2} \hat{k} \tag{4.171}$$

$$\frac{\mathbf{M}_{\bigcirc,d2}}{p_{d2}} = -\frac{r_b^2 \, h_s \, (\theta - \phi_{os} + 2\,\phi_{i0} - \phi_e + 2\,\pi\,N_c - \pi) \, (\theta + \phi_{os} - \phi_e + 2\,\pi\,N_c + \pi)}{2} \hat{k}. \tag{4.172}$$

4.11 Discharge Chamber *dd*

While the suction and compression chambers are relatively straight-forward to treat analytically, the discharge chamber *dd* has problematic geometry. This is due

to the variety of families of curves that can be used to close off the discharge chamber geometry. The goal of this section is to introduce a number of options for closing the discharge region geometry, and provide the necessary analysis to calculate all geometric parameters. As was shown previously in Section 4.2, the scroll wraps are formed of involute curves from the starting involute angle to the ending involute angle. The remaining portion of the scroll is formed by the discharge region curves. There are a number of simple analytic solutions for the discharge geometry, for which the necessary conditions are:

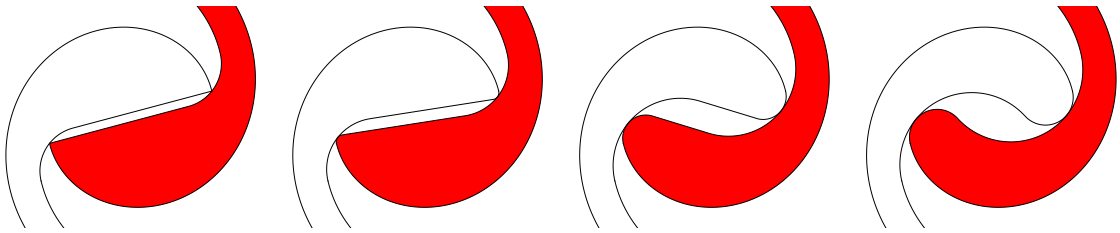
- Curves used must be tangent to the involute at inner starting angle
- Curves used must be tangent to the involute at outer starting angle[2]
- Curves must be selected such that the scrolls do not contact each other during operation
- Curves must pass through the points on the scrolls defined by the inner and outer involute starting angles

Figures 4.22 and 4.23 demonstrate two families of curves which result in solutions that satisfy the constraints listed above. The first family of curves (Figure 4.22) is composed of an arc-line-arc set where the arcs and lines are tangent to each other and the involutes they are connected to. Constraints on the geometry of the curve to avoid collision and yield several special cases will be shown below. The second family of curves (Figure 4.23) which can be used for the discharge region is a set of two arcs which are tangent to each other and the involutes they are connected to. The radius r_{a2} refers to the arc connected to the outer involute of the scroll.

4.11.1 Two Arc Discharge Geometry

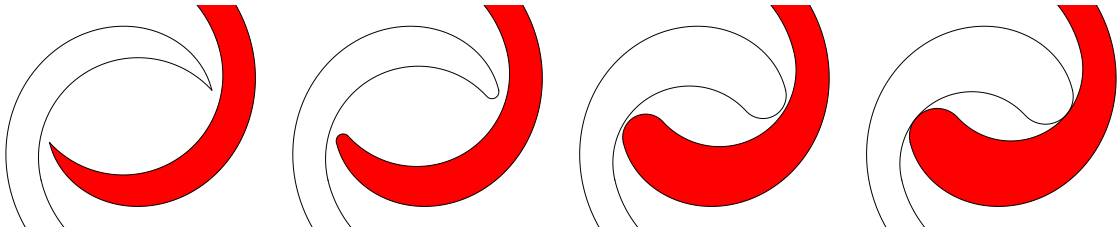
When using the two-arc discharge region geometry, the curves can be defined as shown in Figure 4.24 where there are two arcs, with arc 2 connected to the outer

[2]In the degenerate two-arc case with only one arc, tangency is not required, though locally, the second infinitesimally small arc can be considered to be locally tangent to the outer involute at the outer starting angle



(a) r_{a2}=0 (b) r_{a2}=$r_{a2,max}$/4 (c) r_{a2}=$r_{a2,max}$/1.2 (d) r_{a2}=$r_{a2,max}$

Figure 4.22. Arc-line-arc options for defining chamber *dd*.



(a) r_{a2}=0 (b) r_{a2}=$r_{a2,max}$/4 (c) r_{a2}=$r_{a2,max}$/1.2 (d) r_{a2}=$r_{a2,max}$

Figure 4.23. Two-tangent arc options for defining chamber *dd*.

involute, and arc 1 connected to the inner involute. A family of solutions can be found based on the radius of arc 2, spanning the range of geometries shown in Figure 4.23. At either the inner or outer starting angle, the arc and the involute are tangent and coincident. This means that at this point, the normal vectors of both the arc and the involute are co-linear, and as a consequence, the center point of the arc lies along the line defined by the unit normal vector at the starting angle of the involute. For both inner and outer involutes, the unit normal vector is defined to point towards the involute (see Section 4.5.1). Thus, the coordinates of the center of each arc are given by

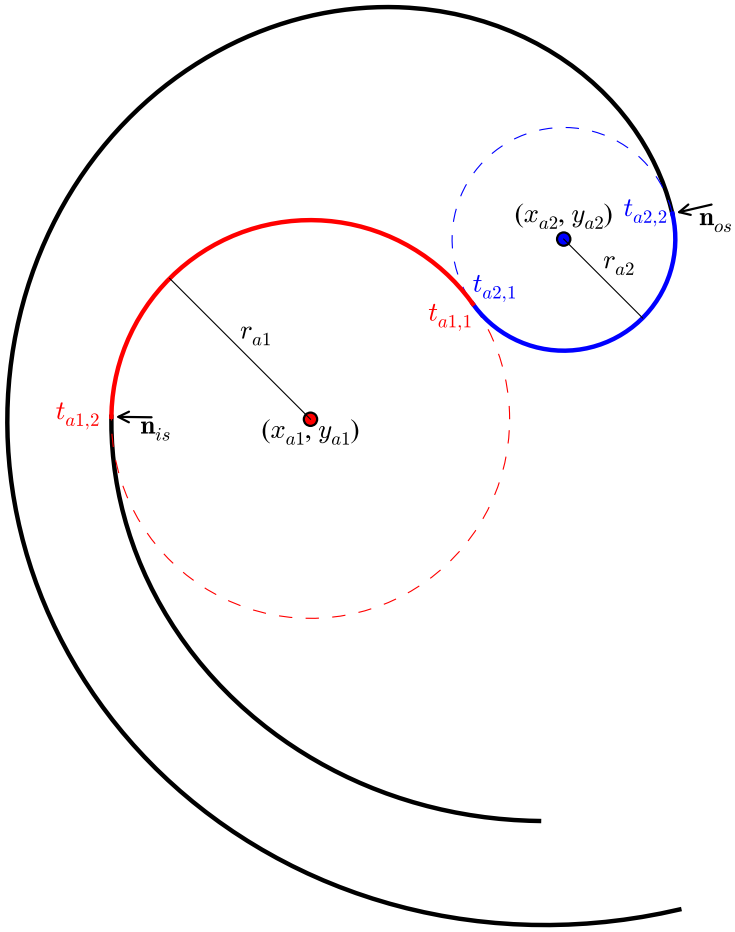


Figure 4.24. Definition of two-arc discharge geometry.

$$x_{a1} = x_{fis} - \sin(\phi_{is}) r_{a1} \tag{4.173}$$

$$y_{a1} = y_{fis} + \cos(\phi_{is}) r_{a1} \tag{4.174}$$

$$x_{a2} = x_{fos} - \sin(\phi_{os}) r_{a2} \tag{4.175}$$

$$y_{a2} = y_{fos} + \cos(\phi_{os}) r_{a2}. \tag{4.176}$$

For a given r_{a2}, the radius of arc 1 can be found to yield tangency between the two arcs. In order to enforce tangency, the distance between the centers of arc 1 and arc 2

must be equal to the sum of the radii of arcs 1 and 2. Thus, expressed mathematically, the equation to be solved is

$$\sqrt{(x_{a1} - x_{a2})^2 + (y_{a1} - y_{a2})^2} = r_{a1} + r_{a2}. \tag{4.177}$$

After simplification and grouping of terms, the result for the radius of arc 1 is

$$r_{a1} = \frac{\frac{1}{2}(\Delta y)^2 + \frac{1}{2}(\Delta x)^2 + r_{a2}\Delta x \sin(\phi_{os}) - r_{a2}\Delta y \cos(\phi_{os})}{r_{a2}\cos(\phi_{os} - \phi_{is}) + \Delta x \sin(\phi_{is}) - \Delta y \cos(\phi_{is}) + r_{a2}} \tag{4.178}$$

with $\Delta x = x_{fis} - x_{fos}$ and $\Delta y = y_{fis} - y_{fos}$. The degenerate case of this set of curves occurs when $r_{a2} = 0$, which yields a single arc with radius

$$r_{a1} = \frac{\frac{1}{2}(\Delta y)^2 + \frac{1}{2}(\Delta x)^2}{\Delta x \sin(\phi_{is}) - \Delta y \cos(\phi_{is})} \text{ (one arc)} \tag{4.179}$$

which is shown in Figure 4.23(a).

The angles $t_{a1,1}$, $t_{a1,2}$, $t_{a2,1}$, and $t_{a2,2}$ can then be obtained with the convention that proceeding along the arc from t_1 to t_2 yields a counter-clockwise traversal. Initial values for the angles are given by

$$t_{a1,1} = \text{atan2}(y_{a2} - y_{a1}, x_{a2} - x_{a1}) \tag{4.180}$$

$$t_{a1,2} = \text{atan2}(y_{is} - y_{a1}, x_{is} - x_{a1}) \tag{4.181}$$

$$t_{a2,1} = \text{atan2}(y_{a1} - y_{a2}, x_{a1} - x_{a2}) \tag{4.182}$$

$$t_{a2,2} = \text{atan2}(y_{os} - y_{a2}, x_{os} - x_{a2}) \tag{4.183}$$

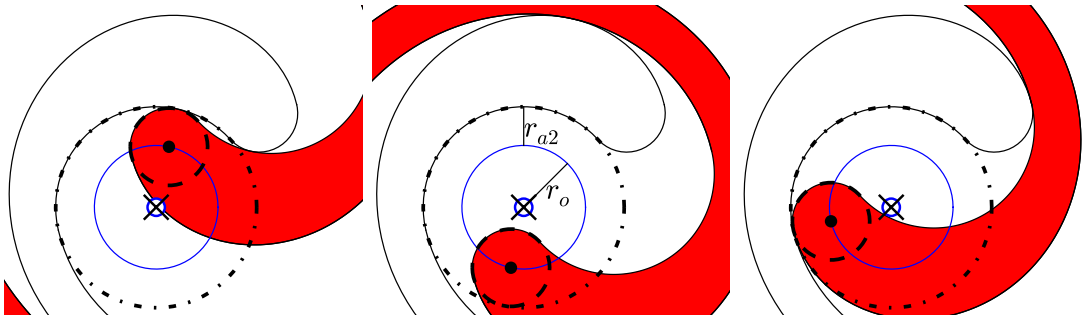
where the function $\text{atan2}(y, x)$ returns the four-quadrant arctangent angle in the range $-\pi$ to π. In order to ensure that the arcs are traversed in a counter-clockwise fashion, the values for $t_{a1,2}$ and $t_{a2,2}$ are increased by increments of 2π until they are greater than $t_{a1,1}$ and $t_{a2,1}$ respectively

In order to extend the compression process, yield a larger effective built-in volume ratio, and reduce the re-expansion losses at the end of the discharge process, it can be advantageous to have a perfect meshing between the two scrolls throughout the discharge process. This is possible using the arc-line-arc method shown in the next

section, but it is also possible to achieve the same effect with two arcs. In order to achieve perfect meshing with two arcs, one necessary condition is that

$$\phi_{is} = \phi_{os} + \pi. \tag{4.184}$$

If this condition is not satisfied, the center of arc 1 and the center of orbit of arc 2 are not coincident because the orbiting scroll lifts off the fixed scroll before the innermost conjugate point reaches the inner starting angle. As a result, it is impossible to construct an arc that maintains contact for $\phi_{os} + \pi > \phi_{is}$. Figures 4.25 and 4.26 show the discharge geometry with and without the condition $\phi_{is} = \phi_{os} + \pi$.



(a) θ=0 (b) $\theta = \pi$ (c) θ=θ_d

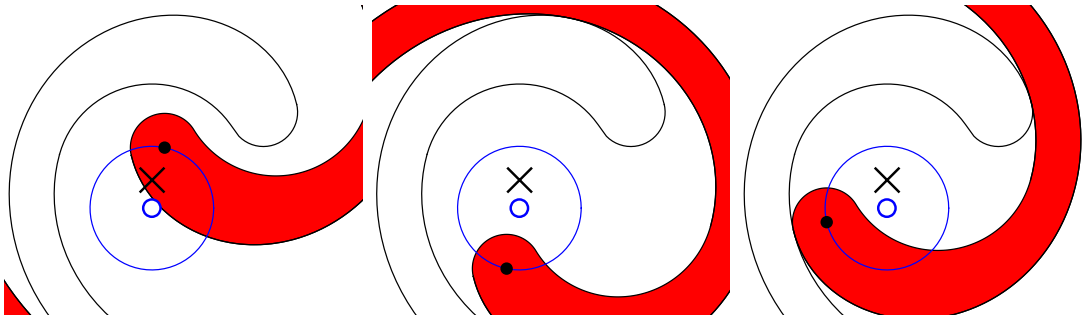
Figure 4.25. Geometry for $\phi_{is} = \phi_{os} + \pi$.

If the perfect meshing pre-condition $\phi_{is} = \phi_{os} + \pi$ is met, it is possible to find an arc that maintains contact between the arcs. The radius of arc 1 is therefore equal to

$$r_{a1} = r_{a2} + r_o. \tag{4.185}$$

There is a unique solution for this condition, which is obtained by equating Eqns. (4.178) and (4.185) and solving for r_{a2}, which yields the solution

$$r_{a2,max} = \frac{1}{2} \frac{-[2\Delta x \sin(\phi_{is}) r_o - 2\Delta y \cos(\phi_{is}) r_o - (\Delta y)^2 - (\Delta x)^2]}{-\Delta x(\sin\phi_{os} - \sin\phi_{is}) + \Delta y(\cos\phi_{os} - \cos\phi_{is})}. \tag{4.186}$$



(a) θ=0 (b) $\theta = \pi$ (c) θ=θ_d

Figure 4.26. Geometry for $\phi_{is} = \phi_{os} + \pi - 0.5$.

This is the largest value of r_{a2} that is possible without scroll-scroll collision, and in addition, it is the radius of r_{a2} for the arc-line-arc solution which corresponds to a line length of zero. If $\phi_{os} + \pi > \phi_{is}$ it is still possible to solve for r_{a2}, but perfect meshing is not achieved. Thus, the maximal value of r_{a2} for $\phi_{os} + \pi > \phi_{is}$ is given by

$$r_{a2,max} = \frac{-\varpi_b + \sqrt{\varpi_b^2 - 4\varpi_a\varpi_c}}{2\varpi_a} \tag{4.187}$$

with

$$\varpi_a = \cos(\phi_{os} - \phi_{is}) + 1 \tag{4.188}$$

$$\varpi_b = r_o\varpi_a - \Delta x(\sin\phi_{os} - \sin\phi_{is}) + \Delta y(\cos\phi_{os} - \cos\phi_{is}) \tag{4.189}$$

$$\varpi_c = \frac{1}{2}\left[2\Delta x\sin(\phi_{is})r_o - 2\Delta y\cos(\phi_{is})r_o - (\Delta y)^2 - (\Delta x)^2\right]. \tag{4.190}$$

4.11.2 Arc-Line-Arc Discharge Geometry

As with the two arc solution to the discharge geometry, for the arc-line-arc solution there are a family of solutions based on the radius of the arc connected to the outer involute of the fixed scroll (r_{a2}). Alternatively, the family could be defined based on the radius of the arc connected to the inner involute of the fixed scroll. Also, just as

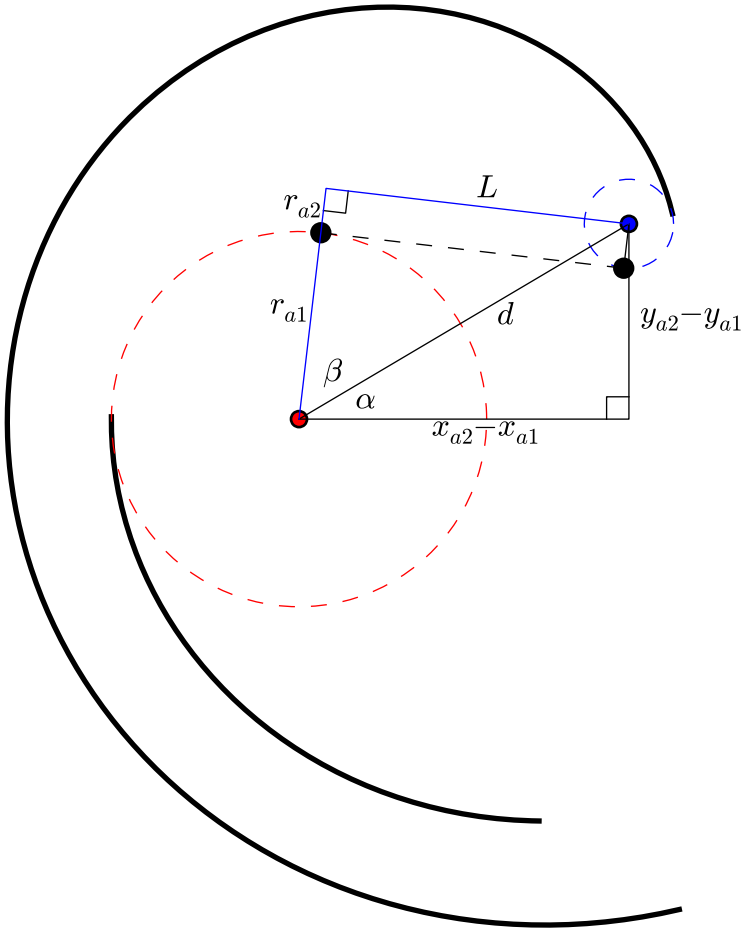


Figure 4.27. Definition of geometric parameters for a discharge region with the arc-line-arc set of curves.

the two-arc solution gives a maximum possible value for r_{a2}, the arc-line-arc shares the same maximum value, in this case, it is the degenerate case where the length of the line connecting the arcs goes to zero, as in Figure 4.22(d). In order to obtain the discharge region geometry, the radii of the two arcs must be determined, where the radius of r_{a2} is between 0.0 and $r_{a2,max}$ from Eqn. (4.186). In order to avoid collision between the scroll wraps, $r_{a1} \geq r_{a2} + r_o$. If $r_{a1} = r_{a2} + r_o$ and $\phi_{os} = \phi_{is} + \pi$, the perfect-meshing-profile is obtained for the arc-line-arc geometry as seen in Figure 4.22. The line connecting the two arcs which is tangent to both arcs can be obtained

with the geometric analysis presented here. As seen in Figure 4.27, the value of angle α can be obtained from

$$\tan\alpha = \frac{y_{a2} - y_{a1}}{x_{a2} - x_{a1}} \tag{4.191}$$

as long as $x_{a2} - x_{a1} \neq 0$. The coordinates of the centers of both arcs are obtained by the same method as for the two-arc solution and the value of β can be obtained from

$$\cos\beta = \frac{r_{a1} + r_{a2}}{d} \tag{4.192}$$

where d is defined by

$$d = \sqrt{(x_{a2} - x_{a1})^2 + (y_{a2} - y_{a1})^2}. \tag{4.193}$$

The coordinates of the point on arc 1 at the location of the tangent line are therefore given by

$$(x_{a1,t}, y_{a1,t}) = (x_{a1} + r_{a1}\cos(\beta + \alpha), y_{a1} + r_{a1}\sin(\beta + \alpha)). \tag{4.194}$$

The length L is given by

$$L = \sqrt{d^2 - (r_{a1} + r_{a2})^2} \tag{4.195}$$

which yields the intersection point on arc 2 of

$$(x_{a2,t}, y_{a2,t}) = (x_{a1,t} + L\sin(\beta + \alpha), y_{a1,t} - L\cos(\beta + \alpha)). \tag{4.196}$$

The slope and intercept of the line segment with coordinates

$$y = m_l x + b_l \tag{4.197}$$

are given by

$$m_l = -\cot(\beta + \alpha) \tag{4.198}$$

$$b_l = y_{a1,t} - m_l x_{a1,t} \tag{4.199}$$

where x takes on the values in the range $x_{a1,t}$ to $x_{a2,t}$. For the degenerate case that the length of the line is zero, then m_l is set to zero and b_l=$y_{a1,t}$. The angles along the

arcs are obtained by a similar method to that of the 2 Arc solution. The angles are given by

$$t_{a1,1} = \beta + \alpha \tag{4.200}$$

$$t_{a1,2} = \text{atan2}(y_{is} - y_{a1}, x_{is} - x_{a1}) \tag{4.201}$$

$$t_{a2,1} = \text{atan2}(y_{a1,t} - y_{a2}, x_{a1,t} - x_{a2}) \tag{4.202}$$

$$t_{a2,2} = \text{atan2}(y_{os} - y_{a2}, x_{os} - x_{a2}) \tag{4.203}$$

and as with the two-arc solution, the values for $t_{a1,2}$ and $t_{a2,2}$ are increased by the value 2π until they are greater than $t_{a1,1}$ and $t_{a2,1}$ respectively

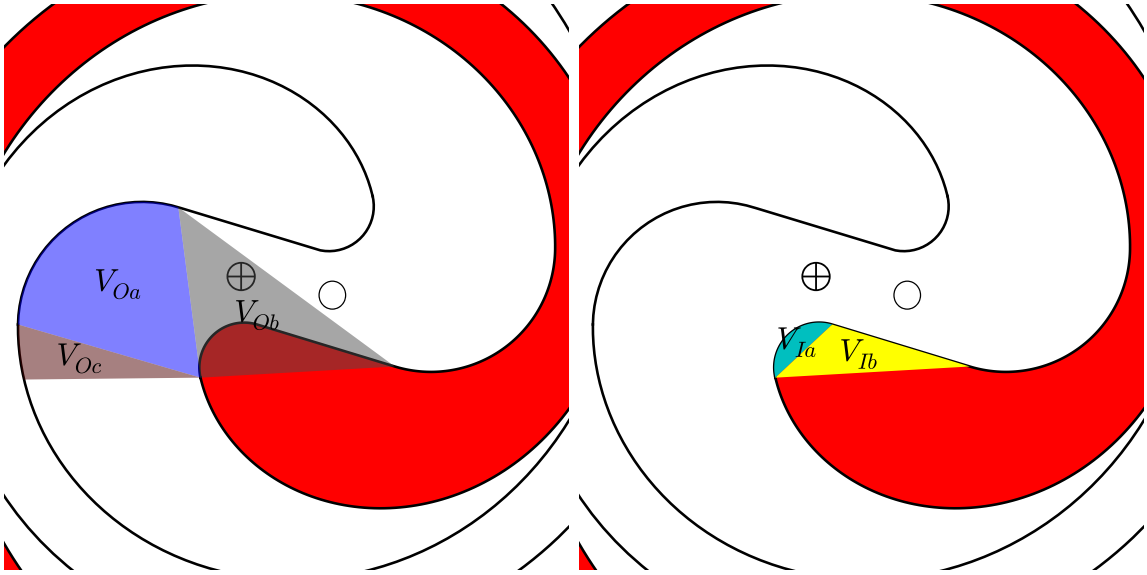
4.11.3 Calculations For *dd* Chamber

In order to calculate the volume of the discharge region, a few important comments are required. Firstly, since the point halfway between the fixed origin and the origin for the orbiting scroll is the point of symmetry of the scroll set, the centroid of the V_{dd} chamber is simply equal to

$$c_{x,dd} = \frac{r_o}{2}\cos\theta_m \tag{4.204}$$

$$c_{y,dd} = \frac{r_o}{2}\sin\theta_m. \tag{4.205}$$

Secondly, from the standpoint of the calculation of the discharge chamber geometry, the two-arc solution can be considered as a degenerate case of the arc-line-arc solution with a line length of 0. Furthermore, if a single arc is used, the degenerate case of the two-arc solution is obtained, which still satisfies the analysis employed for the arc-line-arc solution. Therefore, the arc-line-arc solution can be used for all possibilities for the discharge chamber. As with the other chambers, the volume of V_{dd} is decomposed into regions with simple analytic forms, but here, one half of the V_{dd} chamber is analyzed.



(a) Outer volume (b) Inner volume

Figure 4.28. Decomposition of one half of V_{dd} chamber (labels of volumes placed at centroid of volume).

In order to calculate the volume of $V_{Oa,dd}$, a slightly different approach is used. From the analysis in Section 4.5.1 with the coordinates of a point on the arc given by

$$x(t) = x_{a1} + r_{a1}\cos t \tag{4.206}$$

$$y(t) = y_{a1} + r_{a1}\sin t \tag{4.207}$$

and the point of integration taken to be the outer involute starting angle of the orbiting scroll with coordinates

$$x_{oos} = -r_b\cos\phi_{oos} - r_b(\phi_{oos} - \phi_{o0})\sin\phi_{oos} + r_o\cos\theta_m \tag{4.208}$$

$$y_{oos} = -r_b\sin\phi_{oos} + r_b(\phi_{oos} - \phi_{o0})\cos\phi_{oos} + r_o\sin\theta_m \tag{4.209}$$

the cross-sectional area is obtained from

$$V_{Oa,dd} = \frac{h_s}{2}\int_{t_{a1,1}}^{t_{a1,2}}\left[(x - x_{oos})\frac{dy}{dt} - (y - y_{oos})\frac{dx}{dt}\right]dt \tag{4.210}$$

and the volume $V_{Oa,dd}$ is therefore

$$V_{Oa,dd} = \frac{h_s r_a}{2}\left[\begin{array}{c}(y_{oos}-y_a)\left(\cos t_{a1,2}-\cos t_{a1,1}\right)-(x_{oos}-x_a)\left(\sin t_{a1,2}-\sin t_{a1,1}\right)\\ +r_{a,1}(t_{a1,2}-t_{a1,1})\end{array}\right] \tag{4.211}$$

and its derivative with respect to the crank angle is equal to

$$\frac{dV_{Oa,dd}}{d\theta} = \frac{h_s r_a r_o}{2}\left[-\cos\theta_m(\cos t_{a1,2}-\cos t_{a1,1})-\sin\theta_m(\sin t_{a1,2}-\sin t_{a1,1})\right] \tag{4.212}$$

where care must be taken to also consider the derivatives of x_{oos} and y_{oos} with respect to the crank angle. The volume $V_{Ob,dd}$ is triangular in cross-section, with one point at (x_{oos}, y_{oos}) and the other two points at an arc angle of $t_{a1,1}$ on both the fixed and orbiting scrolls. The points on the fixed scroll are given by

$$x_{l,1} = x_{a1,1} + r_{a1}\cos t_{a1,1} \tag{4.213}$$

$$y_{l,1} = y_{a1,1} + r_{a1}\sin t_{a1,1} \tag{4.214}$$

and the points on the orbiting scroll are the points from Eqn. (4.213) mirrored through the point $(\frac{r_o}{2}\cos\theta_m,\ \frac{r_o}{2}\sin\theta_m)$. This yields the volume for the segment of

$$V_{Ob,dd} = \frac{h_s}{2}\left[(r_o x_{oos}-r_o x_{1,l})\sin\theta_m-(\cos\theta_m r_o-2x_{1,l})\,y_{oos}+y_{1,l}(r_o\cos\theta_m-2x_{oos})\right] \tag{4.215}$$

$$\frac{dV_{Ob,dd}}{d\theta} = \frac{r_o h_s}{2}\left(r_o-y_{oos}\sin\theta_m-x_{oos}\cos\theta_m-y_{1,l}\sin\theta_m-x_{1,l}\cos\theta_m\right). \tag{4.216}$$

The volume $V_{Oc,dd}$ is formed by the remaining part of the involute between the involute angles ϕ_{is} and $\phi_{os}+\pi$. The integral is taken around the point at (x_{oos}, y_{oos}) yielding

$$V_{Oc,dd} = \frac{h_s r_b}{6}\left(\begin{array}{c}3\,r_o\ (\phi_{os}-\phi_{i0}+\pi)\ \sin(\theta+\phi_{os}-\phi_e)+3\,r_o\cos(\theta+\phi_{os}-\phi_e)\\ +3r_o\ (\phi_{is}-\phi_{i0})\ \sin(\theta+\phi_{is}-\phi_e)+3\,r_o\cos(\theta+\phi_{is}-\phi_e)\\ +3\,r_b\ [(\phi_{is}-\phi_{i0})\ (\phi_{os}-\phi_{o0})+1]\ \sin(\phi_{os}-\phi_{is})\\ -3\,r_b\ (\phi_{os}-\phi_{o0}-\phi_{is}+\phi_{i0})\ \cos(\phi_{os}-\phi_{is})\\ +r_b\left((\phi_{os}+\pi-\phi_{i0})^3-(\phi_{is}-\phi_{i0})^3\right)+3r_o\end{array}\right) \tag{4.217}$$

$$\frac{dV_{Oc,dd}}{d\theta} = \frac{h_s r_b r_o}{2} \left(\begin{array}{c} (\phi_{os} - \phi_{i0} + \pi)\cos(\theta + \phi_{os} - \phi_e) - \sin(\theta + \phi_{os} - \phi_e) \\ +(\phi_{is} - \phi_{i0})\cos(\theta + \phi_{is} - \phi_e) - \sin(\theta + \phi_{is} - \phi_e) \end{array} \right). \tag{4.218}$$

The volume $V_{Ia,dd}$ is an arc in cross-section, and a polar integration is taken around the point at the outer involute starting angle with coordinates from Eqn. (4.208). The outer surface of the volume is the arc 2 of the scroll. Thus, the coordinates of a point on the arc are given by

$$x(t) = -r_{a2}\cos(t) + r_o\cos\theta_m \tag{4.219}$$

$$y(t) = -r_{a2}\sin(t) + r_o\sin\theta_m \tag{4.220}$$

where t takes on the values in the range $t_{a2,1}$ to $t_{a2,2}$. Carrying out a polar integration yields the solution

$$V_{Ia,dd} = \frac{h_s r_{a2}}{2} \left[\begin{array}{c} x_{a2}(\sin t_{a2,2} - \sin t_{a2,1}) - y_{a2}(\cos t_{a2,2} - \cos t_{a2,1}) \\ -r_b[\sin(t_{a2,2} - \phi_{os}) - \sin(t_{a2,1} - \phi_{os})] \\ -r_b(\phi_{os} - \phi_{o0})[\cos(t_{a2,2} - \phi_{os}) - \cos(t_{a2,1} - \phi_{os})] \\ +r_{a2}(t_{a2,2} - t_{a2,1}) \end{array} \right] \tag{4.221}$$

with derivative with respect to the crank angle of

$$\frac{dV_{Ia,dd}}{d\theta} = 0 \tag{4.222}$$

Volume $V_{Ib,dd}$ is also triangular in cross-section, with one vertex at the point (x_{oos}, y_{oos}) and the other two at the ends of the line segment on the orbiting scroll, if it exists. If the line segment doesn't exist, the volume of $V_{Ib,dd}$ is equal to 0.

$$V_{Ib,dd} = \frac{-h_s(x_{2,l} - x_{1,l})}{2} \left[\begin{array}{c} r_b m_l[\cos\phi_{os} + (\phi_{os} - \phi_{o0})\sin\phi_{os}] + b_l \\ -r_b[\sin\phi_{os} - (\phi_{os} - \phi_{o0})\cos\phi_{os}] \end{array} \right] \tag{4.223}$$

where the coordinates of $x_{2,l}$ and $x_{1,l}$ are given by

$$x_{l,1} = x_{a1,1} + r_{a1}\cos t_{a1,1} \tag{4.224}$$

$$x_{2,1} = x_{a2,1} + r_{a2}\cos t_{a2,1} \tag{4.225}$$

and the intercept of the line (b_l) is given by

$$b_l = y_{a1,1} + r_{a1} \sin t_{a1,1} - m_l(x_{a1,1} + r_{a1} \cos t_{a1,1}) \tag{4.226}$$

which yields the derivative of the volume of

$$\frac{dV_{Ib,dd}}{d\theta} = 0. \tag{4.227}$$

The volumes of the constituent parts for the chamber *dd* can be summarized by the Table 4.1.

Table 4.1 Definition of terms for differing discharge geometries.

Type	$V_{Oa,dd}$	$V_{Ob,dd}$	$V_{Ia,dd}$	$V_{Ib,dd}$
Arc-Line-Arc	$\neq 0$	$\neq 0$	$\neq 0$	$\neq 0$
2 Arc	$\neq 0$	0	$\neq 0$	0
1 Arc	$\neq 0$	0	0	0

Thus, the total volume of the V_{dd} chamber can be given by

$$V_{dd} = 2\left[V_{Oa,dd} + V_{Ob,dd} - V_{Ia,dd} - V_{Ib,dd}\right] \tag{4.228}$$

and the derivative of the volume with respect to the crank angle by

$$\frac{dV_{dd}}{d\theta} = 2\left[\frac{dV_{Oa,dd}}{d\theta} + \frac{dV_{Ob,dd}}{d\theta} - \frac{dV_{Ia,dd}}{d\theta} - \frac{dV_{Ib,dd}}{d\theta}\right]. \tag{4.229}$$

Forces

The calculations for the forces in the discharge chamber *dd* are simplified by the fact that the discharge chamber is composed of arcs and a line segment. The normal vector for arc 1 which points in the direction of the orbiting scroll surface is given by

$$\mathbf{n}_x = -\cos t \tag{4.230}$$

$$\mathbf{n}_y = -\sin t. \tag{4.231}$$

The differential area of the arc is equal to $dA = h_s r_{a1} dt$ where t is the parameter of the arc and thus, the integration of the product $\mathbf{n}dA$ yields the force terms

$$\frac{\mathbf{F}_{x,dd,a1}}{p_{dd}} = -h_s r_{a1} \left[\sin(t_{a1,2}) - \sin(t_{a1,1})\right] \tag{4.232}$$

$$\frac{\mathbf{F}_{y,dd,a1}}{p_{dd}} = +h_s r_{a1} \left[\cos(t_{a1,2}) - \cos(t_{a1,1})\right]. \tag{4.233}$$

The moment around the crank pin (whose location is given by Eqn. (4.40)) generated by arc 1 is given by

$$\frac{\mathbf{M}_{\bigcirc,dd,a1}}{p_{dd}} = -r_{a1}\, h_s\, \left[\left(\sin\left(t_{a1,2}\right) - \sin\left(t_{a1,1}\right)\right)\, y_{a1} + \left(\cos\left(t_{a1,2}\right) - \cos\left(t_{a1,1}\right)\right)\, x_{a1}\right]. \tag{4.234}$$

The analysis for arc 2 proceeds in the same fashion except that the normal vector has the opposite direction so that it points towards the center of the arc. Thus, the force components for arc 2 are given by

$$\frac{\mathbf{F}_{x,dd,a2}}{p_{dd}} = h_s r_{a2} \left[\sin(t_{a2,2}) - \sin(t_{a2,1})\right] \tag{4.235}$$

$$\frac{\mathbf{F}_{y,dd,a2}}{p_{dd}} = -h_s r_{a2} \left[\cos(t_{a2,2}) - \cos(t_{a2,1})\right]. \tag{4.236}$$

If arc 2 does not exist ($r_{a2} = 0$), then the arc angles ($t_{a2,2}$ & $t_{a2,1}$) are set to be equal, yielding zero force components for arc 2. The moment around the crank pin (whose location is given by Eqn. (4.40)) generated by arc 2 is given by

$$\frac{\mathbf{M}_{\bigcirc,dd,a2}}{p_{dd}} = r_{a2}\, h_s\, \left[\left(\sin\left(t_{a2,2}\right) - \sin\left(t_{a2,1}\right)\right)\, y_{a2} + \left(\cos\left(t_{a2,2}\right) - \cos\left(t_{a2,1}\right)\right)\, x_{a2}\right]. \tag{4.237}$$

For the line, the end points of the line are given by the coordinates

$$(x_{a1,t}) = (-x_{a1} - r_{a1}\cos t_{a1,1} + r_o \cos\theta_m) \tag{4.238}$$

$$(y_{a1,t}) = (-y_{a1} - r_{a1}\sin t_{a1,1} + r_o \sin\theta_m) \tag{4.239}$$

$$(x_{a2,t}) = (-x_{a2} - r_{a2}\cos t_{a2,1} + r_o \cos\theta_m) \tag{4.240}$$

$$(y_{a2,t}) = (-y_{a2} - r_{a2}\sin t_{a2,1} + r_o \sin\theta_m). \tag{4.241}$$

The length of the line passing from arc 1 to arc 2 is given by

$$L = \sqrt{(x_{a1,t} - x_{a2,t})^2 + (y_{a1,t} - y_{a2,t})^2} \tag{4.242}$$

and a unit tangent vector pointing along the line from arc 1 to arc 2 can be given by

$$\mathbf{L} = \frac{x_{a2,t} - x_{a1,t}}{L}\hat{i} + \frac{y_{a2,t} - y_{a1,t}}{L}\hat{j}. \tag{4.243}$$

The unit normal vector pointing towards the scroll surface for the linear segment must then satisfy the relationship

$$\mathbf{n} \cdot \mathbf{L} = 0 \tag{4.244}$$

because the vectors are orthogonal, which yields the result for the normal vector components of

$$\mathbf{n} = -\frac{1}{\sqrt{1 + \frac{\mathbf{L}_x^2}{\mathbf{L}_y^2}}}\hat{i} + \frac{\mathbf{L}_x/\mathbf{L}_y}{\sqrt{1 + \frac{\mathbf{L}_x^2}{\mathbf{L}_y^2}}}\hat{j}. \tag{4.245}$$

There are two solutions; for one solution the normal vector points towards the scroll wrap, and for the other solution, the normal vector points away from the scroll wrap. The correct solution can be found by ensuring that

$$\mathbf{L} \times \mathbf{n} > 0 \tag{4.246}$$

which enforces that the cross product of the vectors is positive, and the normal vector points towards the scroll surface. The force components from the line are therefore given by

$$\frac{\mathbf{F}_{x,dd,line}}{p_{dd}} = h_s \mathbf{n}_x L \tag{4.247}$$

$$\frac{\mathbf{F}_{y,dd,line}}{p_{dd}} = h_s \mathbf{n}_y L. \tag{4.248}$$

The moment around the crank pin generated by the line segment is given by

$$\frac{\mathbf{M}_{\bigcirc,dd,line}}{p_{dd}} = \mathbf{r}_{\bigcirc,dd,line} \times \frac{\mathbf{F}_{dd,line}}{p_{dd}} \tag{4.249}$$

where $\mathbf{r}_{\bigcirc,dd,line}$ is the vector from ($x_{\bigcirc}$, $y_{\bigcirc}$) to the midpoint of the line segment.

If there is no line segment, the ends of the arcs in the discharge region are coincident, and there is no applied gas force. If the line segment is vertical, the length vector component $\mathbf{L}_x$ is zero, and the normal vector is equal to

$$\mathbf{n} = -\hat{i}. \tag{4.250}$$

Depending on the orientation of the vertical line, it may be required to flip the sign of the normal vector, the proper sign yields $\mathbf{L} \times \mathbf{n} > 0$. There is a small segment of the inner involute of the orbiting scroll between the involute angles of ϕ_{is} and $\phi_{os} + \pi$ which also is part of the discharge chamber. Thus, the force components of this part can be obtained from

$$\frac{\mathbf{F}_{dd,involute}}{p_{dd}} = -h_s r_b \int_{\phi_{is}}^{\phi_{os}+\pi} \left[(\phi - \phi_{i0}) \sin(\phi) \hat{i} + (\phi - \phi_{i0}) \cos(\phi) \hat{j} \right] d\phi \tag{4.251}$$

where the solution for the force terms are in Appendix B.4. The moment around the crank pin for this involute segment is equal to

$$\frac{\mathbf{M}_{\bigcirc,dd,involute}}{p_{dd}} = -\frac{h_s r_b^2 \ (\phi_{os} - \phi_{is} + \pi) \ (\phi_{os} + \phi_{is} - 2\,\phi_{i0} + \pi)}{2}. \tag{4.252}$$

The total force components from the constituent curves in the *dd* chamber are given by

$$\frac{\mathbf{F}_{x,dd}}{p_{dd}} = \frac{\mathbf{F}_{x,dd,line}}{p_{dd}} + \frac{\mathbf{F}_{x,dd,a1}}{p_{dd}} + \frac{\mathbf{F}_{x,dd,a2}}{p_{dd}} + \frac{\mathbf{F}_{x,dd,involute}}{p_{dd}} \tag{4.253}$$

$$\frac{\mathbf{F}_{y,dd}}{p_{dd}} = \frac{\mathbf{F}_{y,dd,line}}{p_{dd}} + \frac{\mathbf{F}_{y,dd,a1}}{p_{dd}} + \frac{\mathbf{F}_{y,dd,a2}}{p_{dd}} + \frac{\mathbf{F}_{y,dd,involute}}{p_{dd}} \tag{4.254}$$

$$\frac{\mathbf{M}_{\bigcirc,dd}}{p_{dd}} = \frac{\mathbf{M}_{\bigcirc,dd,line}}{p_{dd}} + \frac{\mathbf{M}_{\bigcirc,dd,a1}}{p_{dd}} + \frac{\mathbf{M}_{\bigcirc,dd,a2}}{p_{dd}} + \frac{\mathbf{M}_{\bigcirc,dd,involute}}{p_{dd}} \tag{4.255}$$

where some of the force component terms may be zero depending on the scroll geometry.

4.12 Flow Areas For Leakage And Primary Flow Paths

Halm (1997) analytically calculated the leakage lengths for the radial and flank leakages, but did not take into account which control volume was on either side of the flow path. Wang et al. (2005) noted this shortcoming of Halm's model and proposed a multiple-segment leakage length model. While Wang's model is an improvement, it still does not completely resolve the leakage length calculations. Blunier (2006; 2009) proposed a similar method to Wang based on a segmented model. The primary problem with the existing leakage length calculations is that they are not sufficiently generic to handle an arbitrary number of compression chambers.

4.12.1 Radial Leakages

In order to calculate the radial leakage flowrate between a given pair of chambers, it is necessary to calculate the radial leakage area between chambers. For the radial leakage this involves calculating the arc length of the leakage and multiplying by the radial gap width. The arc length of the leakage is defined based on the beginning and ending involute angles which define a line segment which separates two chambers.

The definitions for the leakage areas depend on the number of pairs of compression chambers in existence at a given crank angle. Figure 4.29 shows three different scroll sets with 0 to 2 pairs of compression chambers in existence. Each of the black dots represents a point along the scroll at which the definitions of the upstream and/or downstream chamber changes. Table 4.2 shows the involute angles which form each of the radial flow paths. In the $N_c > 1$ case, α takes on the values from 2 to N_c inclusive. Each of the flow areas are calculated between each of the compression chambers in existence at a given crank angle.

In order to calculate the length of a leakage, the ϕ_{min} and ϕ_{max} values are obtained, and the curve length is obtained from application of

$$s_{radial} = r_b \left(\frac{1}{2}(\phi_{max}^2 - \phi_{min}^2) - \phi_0(\phi_{max} - \phi_{min}) \right) \tag{4.256}$$

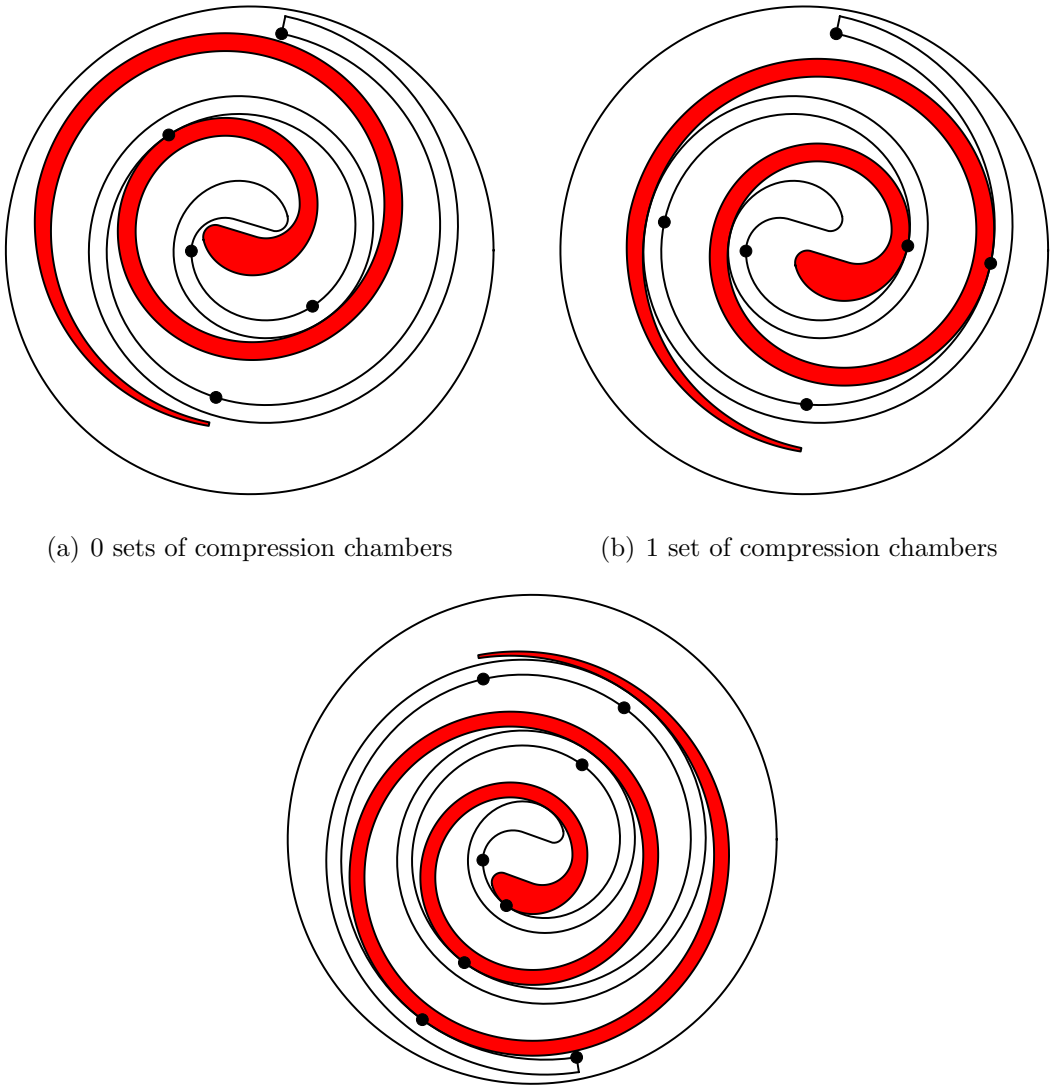
which is the integration of the arclength differential element $ds = r_b(\phi - \phi_0)$. Finally the radial leakage area for each flow path is obtained by multiplying the curve length s_{radial} by the radial gap width δ_{radial}

4.12.2 Flank Leakage

There are only a few points in the compressor that experience flank leakage, and at each flank leakage location, the leakage area is equal to

$$A = h_s \delta_{flank}. \tag{4.257}$$

The gaps that have flank leakage are between:



(a) 0 sets of compression chambers

(b) 1 set of compression chambers

(c) 2 sets of compression chambers

Figure 4.29. Critical involute angles defining radial leakage lengths.

- suction chambers and compression chambers along the same path
- compression chambers and other compression chambers along the same path
- compression chambers and the discharge region along the same path

At each of these locations the flank leakage area is calculated for use in the mass flow model.

Table 4.2 Radial Leakage Angles.

N_c	Chamber Pairs	ϕ_{max}	ϕ_{min}
All	s_2-sa/s_1-sa	ϕ_{ie}	$\max(\phi_{ie}-\theta,\ \phi_{s-sa})$
	s_2-s_1/s_1-s_2	ϕ_{s-sa}	$\min(\phi_{ie}-\theta,\ \phi_{s-sa})$
N_c=0	d_2-s_1/d_1-s_2	$\phi_{ie}-\theta$	$\phi_{ie}-\theta-\pi$
	d_2-d_1/d_1-d_2	$\phi_{ie}-\theta-\pi$	ϕ_{is}
N_c=1	c_2-sa/c_1-sa	$\max(\phi_{ie}-\theta,\ \phi_{s-sa})$	ϕ_{s-sa}
	c_2-s_1/c_1-s_2	$\max(\phi_{ie}-\theta-\pi,\ \phi_{s-sa})$	$\phi_{ie}-\theta-\pi$
	c_2-c_1/c_1-c_2	$\phi_{ie}-\theta-\pi$	$\phi_{ie}-\theta-2\pi$
	d_2-c_1/d_1-c_2	$\phi_{ie}-\theta-2\pi$	ϕ_{is}
$N_c>1$	$c_{2,1}$-sa/$c_{1,1}$-sa	$\max(\phi_{ie}-\theta,\ \phi_{s-sa})$	ϕ_{s-sa}
	$c_{2,1}$-s_1/$c_{1,1}$-s_2	$\max(\phi_{ie}-\theta-\pi,\ \phi_{s-sa})$	$\phi_{ie}-\theta-\pi$
	$c_{2,1}$-$c_{1,1}$/$c_{1,1}$-$c_{2,1}$	$\phi_{ie}-\theta-\pi$	$\phi_{ie}-\theta-2\pi$
	$c_{2,\alpha}$-$c_{1,\alpha-1}$/$c_{1,\alpha}$-$c_{2,\alpha-1}$	$\phi_{ie}-\theta-2\pi(\alpha-1)$	$\phi_{ie}-\theta-2\pi(\alpha-1)-\pi$
	$c_{2,\alpha}$-$c_{1,\alpha}$/$c_{1,\alpha}$-$c_{2,\alpha}$	$\phi_{ie}-\theta-2\pi(\alpha-1)-\pi$	$\phi_{ie}-\theta-2\pi\alpha$
	d_2-c_1/d_1-c_2	$\phi_{ie}-\theta-2\pi N_c$	ϕ_{is}

4.12.3 Suction And Discharge Flow Areas

During the course of the suction process, the flow area of the gap between the suction area chamber sa and the suction chambers s_1 and s_2 varies. The flow area is zero at a crank angle of 0, at which point the volume of the suction chambers are also equal to zero. The area of the gap between the sa and s_1 chambers (by symmetry equal to area between sa and s_2 chambers) is given by the distance between the end of the fixed scroll and the point on the outer involute of the orbiting scroll at the involute angle of $\phi_{s,sa}$ times the height of the scroll wrap. This line can be seen in Figure 4.14 between the end of the fixed scroll and the point marked $\square$; the line has end points of

$$x_e = r_b \cos\phi_{ie} + r_b(\phi_{ie} - \phi_{i0}) \sin\phi_{ie} \tag{4.258}$$

$$y_e = r_b \sin\phi_{ie} - r_b(\phi_{ie} - \phi_{i0}) \cos\phi_{ie} \tag{4.259}$$

$$x_{s-sa} = -r_b \cos\phi_{s-sa} - r_b(\phi_{s-sa} - \phi_{o0}) \sin\phi_{s-sa} + r_o \cos\theta_m \tag{4.260}$$

$$y_{s-sa} = -r_b \sin\phi_{s-sa} + r_b(\phi_{s-sa} - \phi_{o0}) \cos\phi_{s-sa} + r_o \sin\theta_m \tag{4.261}$$

which yields a flow area of

$$A_{s-sa} = h_s \sqrt{(x_e - x_{s-sa})^2 + (y_e - y_{s-sa})^2}. \tag{4.262}$$

Halm (1997) presented a simple form for the suction flow area, given by

$$A_{s-sa,Halm} = h_s r_o (1 - \cos\theta) \tag{4.263}$$

and Figure 4.30 shows both Halm's form and the more precise form presented here.

For the discharge chambers, shortly after the discharge angle there is a pressure equilibration process, and the speed of the equilibrium is driven by flow area between the chamber dd connected to the discharge port and the discharge chambers d_1 and d_2 that just came into existence from the innermost compression chambers. There are a number of possible definitions for the flow area between the orbiting scroll and the fixed scroll during the course of this process. The chambers are divided by a line,

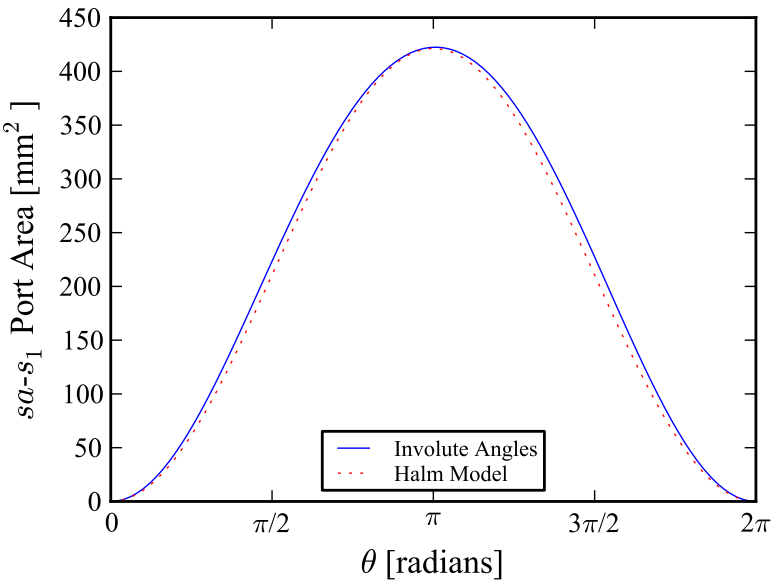
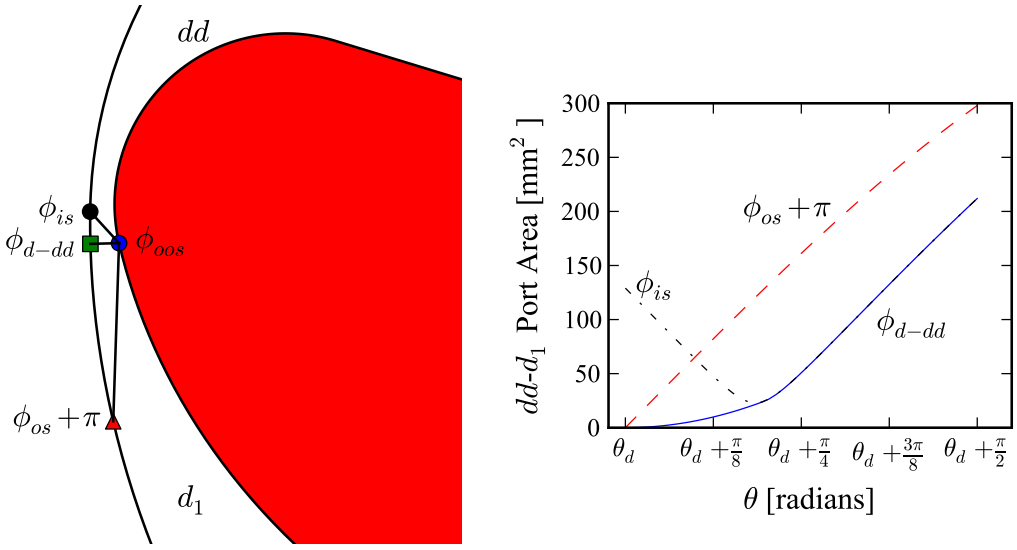


Figure 4.30. Area between chambers sa and s_1 as a function of crank angle.

and one of the points of the line for all the definitions is the starting angle of the outer involute of the orbiting scroll. In the first definition, the second point is at the involute angle $\phi_{os} + \pi$ on the fixed scroll. In this definition, the flow area starts at zero, which is physically correct since right at the discharge angle the scrolls are still in contact. A second definition is that the partner point is the involute angle ϕ_{is}. Finally a third definition is that the partner point is located at the involute angle ϕ_{d-dd} that minimizes the distance between the orbiting scroll and the fixed scroll, and is bound to be ϕ_{is} at minimum to ensure that it stays on the involute portion of the scroll. Figure 4.31(a) shows a schematic of these three definitions shortly after the discharge angle. ϕ_{d-dd} is equal to $\phi_{os} + \pi$ at the discharge angle, and moves towards ϕ_{is} quickly. Figure 4.31(b) shows the areas calculated by these three definitions up to a quarter revolution after the discharge angle which shows that the ϕ_{d-dd} and ϕ_{is} definitions quickly converge, but $\phi_{os} + \pi$ generally overpredicts the area. While there is no exact solution to the definition of the angle to be used to define the flow area between dd and d_1, ϕ_{d-dd} is believed to best capture the geometry of the flow, and

was used in the analysis that follows. In practice, it is possible that the curves of the discharge region might form the limiting flow area, and Figure 4.31(a) suggests that perhaps even for this compressor an alternative definition for the area between the dd and d_1 chambers should be employed. In general, the definition employed here will tend to overpredict the flow area at angles much greater than the discharge angle. Approximately one quarter of a revolution is required for pressures to reach equilibrium in the discharge region.



(a) Schematic for $\theta = \theta_d + \pi/6$

(b) Flow areas for varied definition of involute angle of the partner point

Figure 4.31. Discharge path between dd and d_1 chambers.

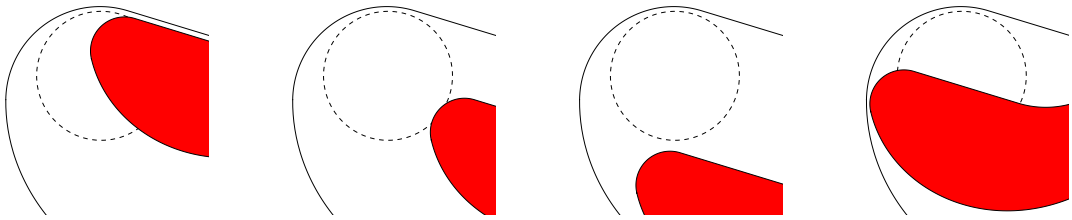
Thus, the coordinates of the point at the involute angle ϕ_{d-dd} on the inner involute of the fixed scroll are given by

$$x_{d-dd} = r_b \cos\phi_{d-dd} + r_b(\phi_{d-dd} - \phi_{i0}) \sin\phi_{d-dd} \tag{4.264}$$

$$y_{d-dd} = r_b \sin\phi_{d-dd} - r_b(\phi_{d-dd} - \phi_{i0}) \cos\phi_{d-dd} \tag{4.265}$$

and the flow area is given by

$$A_{d-dd} = h_s \sqrt{(x_{d-dd} - x_{oos})^2 + (y_{d-dd} - y_{oos})^2} \tag{4.266}$$



(a) θ=0 (b) θ=π/2 (c) θ=π (d) θ=3π/2

Figure 4.32. Discharge port blockage over one rotation.

where the coordinates of the point (x_{oos}, y_{oos}) are given in Eqn. (4.208). The flow area between the chambers dd and d_1 and dd and d_2 are equivalent.

4.12.4 Discharge Port

The flow through the discharge port of the compressor is complicated by the fact that the orbiting scroll partly blocks the discharge port during the course of a rotation. Figure 4.32 shows a discharge port over the course of a rotation. In order to calculate the free area of the port, numerical routines are needed. At a given step of the revolution, two polygons are constructed, one representing the outline of the discharge port, and another representing the tip of the orbiting scroll. The free area of the discharge port is then equal to the total area of the discharge port minus the intersection area of the polygons. Calculating the intersection area of two polygons is a complex mathematical problem for which several algorithms are available. The General Polygon Clipper Library (Murta, 2010) is a powerful and computationally-efficient set of computer code that can calculate the intersection of polygons and return another set of polygons that span the intersection region. The area of each of the intersection polygons can then be obtained from Eqn. (4.44). Thus, the discharge port free area is given by

$$A_{free} = \frac{\pi D_{port}^2}{4} - A_{intersection} \tag{4.267}$$

where D_{port} is the diameter of the port and $A_{intersection}$ is the sum total of the polygons that form the intersection of the port and the tip of the scroll. The analysis here assumes that the port is circular in cross-section, but in general, complex discharge port geometries are possible, in which case the port area would not be that of a circle but would be obtained numerically. The analysis required for the port free area is similar, except that a polygon for the outline of the discharge port must be constructed in several pieces rather than in one step for the circular discharge port.

The entire area of the discharge port is assumed to be connected to the *dd* chamber, and once the discharge chambers have merged, the entire discharge region then communicates with the exhaust of the compressor. The duration of the merging process is relatively short which introduces little error employing the assumption that all the flow area is connected to the *dd* chamber.

The calculation of the discharge port free area is relatively computationally expensive, so in order to minimize the computational overhead, the discharge port free area is calculated for a number of points per rotation at the beginning of the model execution. Quadratic interpolation is then used to find discharge port blockage areas for crank angles in between the sampled values. Figure 4.33 shows the free area of the discharge port over the course of one rotation. The discharge port modeled here is that of the Sanden compressor described in more the next section.

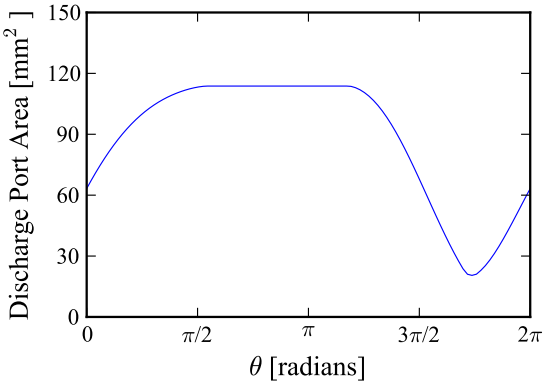


Figure 4.33. Discharge port free area over the course of one rotation.

4.13 Volume Results

In this section, results of the calculations of the chamber volumes for the Sanden scroll compressor are presented. The geometric parameters of this compressor are found in Table 4.3.

Table 4.3 Geometric parameters for Sanden compressor.

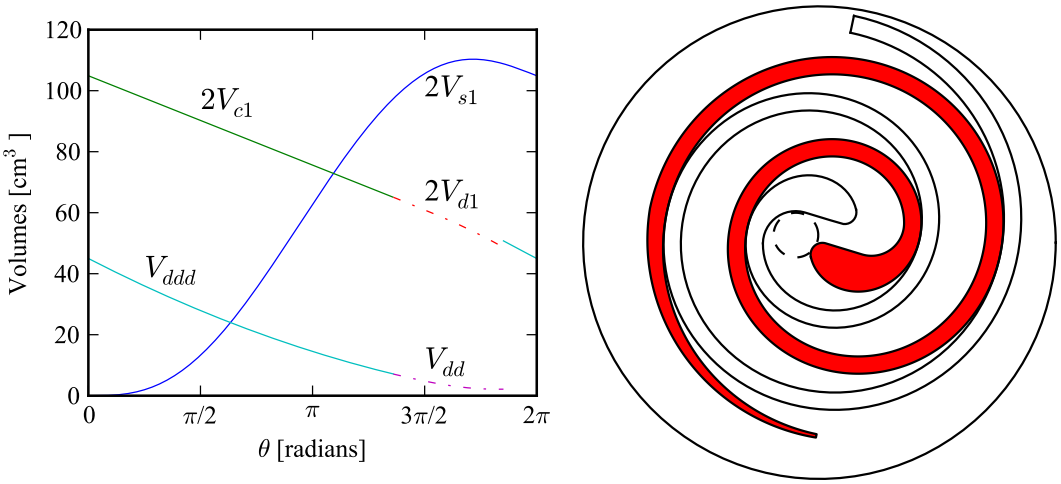
r_b	r_o	ϕ_{i0}	ϕ_{is}	ϕ_{ie}	ϕ_{o0}	ϕ_{os}	ϕ_{oe}	h_s
mm	mm	rad	rad	rad	rad	rad	rad	mm
3.522	6.405	0.1983	4.7	15.5	-1.125	1.8	15.5	32.89

Using these geometric parameters, the volumes of the control volumes over one rotation are shown in Figure 4.34(a). The discharge port and discharge chamber curve geometry is found in Table 4.4, where the coordinates of the discharge port, shown in Figure 4.34(b) nestled in the fixed scroll are based on the fixed scroll coordinate system. The radii for the arc-line-arc set that form the discharge chamber (r_{a1} and r_{a2}) are based on the model presented in Section 4.11.2.

Table 4.4 Discharge geometry for Sanden compressor.

$x_{0,disc}$	$y_{0,disc}$	r_{disc}	Type	r_{a1}	r_{a2}
mm	mm	mm	-	mm	mm
-7.0	-1.1	6.0	Arc-Line-Arc	8.80	3.18

The scroll compressor has an inlet tube of inner diameter 18.8 mm, and discharge tube of inner diameter 16.6 mm. The length of the suction and discharge tubes in the compressor shell are approximately 4 cm. After the flow enters into the compressor, it goes into the shell, which has an internal diameter of 12.30 cm, seen as the outer wall in Figure 4.34(b).



(a) Volumes over one rotation (b) Geometry of scroll compressor

Figure 4.34. The Sanden model TRS-105 compressor used in this study.

It is therefore possible to follow the evolution of the control volumes through the entire compression process. First the suction chamber, which begins with zero volume, begins to open up to the suction port and mixture flows into the suction chamber. The suction chamber continues to increase in size through the rotation until it reaches a point near but not quite at the end of the rotation where it has a maximum in volume. After the maximum it decreases in size slightly to the end of the rotation where the suction chamber gets pinched off and becomes a compression chamber. The volume of the compression chamber decreases linearly over the next part of the rotation until the discharge angle is reached in the second rotation. At the discharge angle, the prior compression chamber is treated as a discharge chamber and the gap between the discharge chamber and the mixture remaining in the discharge region from the previous rotation begins to increase. This allows for pressure equilibration among the discharge chambers and once that is achieved, the chambers are then mixed back into the discharge chamber. The merged discharge chamber continues through the rest of the rotation and through to the discharge angle of the following rotation when it again opens back to equilibrate with a compression chamber.

CHAPTER 5. SCROLL COMPRESSOR OVERALL MODEL

5.1 Motivation

In the previous chapter, a geometric model for the scroll compressor was developed, including volume of all the working chambers, flow areas, and forces and centroids. The analysis in this chapter is used to develop an overall model for the performance of the scroll compressor with oil flooding. Appendix F contains all the code for the scroll compressor geometric and overall modeling.

5.2 Mixture Properties

In order to define mixture properties for the refrigerant-oil mixture it is assumed that the phases are completely separated, and at the same temperature and pressure. The validity of this assumption was discussed above in Section 3.3.1. Therefore, for a homogeneous mixture, the mixture specific volume is given by

$$v_m = x_l v_l + x_g v_g \tag{5.1}$$

where x_l and x_g are the mass fractions of liquid and gas respectively, and $x_g = 1 - x_l$. This formulation is simply a mass-fraction-weighted-average of the individual gas and liquid specific volumes at the overall temperature and pressure. The mixture density ρ_m is just the reciprocal of the mixture specific volume. The mixture specific internal energy u_m, mixture specific enthalpy h_m, mixture specific entropy s_m, and mixture constant pressure specific heat $c_{p,m}$ are also defined as oil mass fraction weighted averages of the properties of the phases.

In order to derive a mixture thermal conductivity, the conductivity is considered as a parallel network of conduction through the gas and liquid. The mixture thermal conductivity is therefore defined as

$$k_m = (1-\alpha)\, k_l + \alpha k_g \tag{5.2}$$

which is a void-fraction weighted average of the conductivities of the gas and liquid individually. Thus the only remaining parameter is the void fraction, the fraction of the mixture's volume that is gas. The gas and liquid phases are assumed to travel at the same speed (the homogeneous flow assumption), and the void fraction α, given by

$$\alpha = \frac{A_g}{A} = \frac{x_g v_g}{x_l v_l + x_g v_g} \tag{5.3}$$

is the fraction of a cross-sectional area that is filled with gas.

A number of different models are available to calculate the viscosity of a mixture, some based on capturing the physically correct values at the all-liquid and all-gas conditions, μ_l and μ_g respectively. One of the models that captures the physically correct boundary values is that of McAdams (1942), for which the mixture viscosity is given by

$$\mu_m = \left(\frac{x_l}{\mu_l} + \frac{x_g}{\mu_g} \right)^{-1} = \frac{\mu_l \mu_g}{\mu_g x_l + \mu_l x_g} \tag{5.4}$$

which by inspection does fulfill the physically correct boundary values. The mixture Prandtl number is given by

$$\mathrm{Pr}_m = \frac{\mu_m c_{p,m}}{k_m} \tag{5.5}$$

where the transport properties are based on mixture transport properties.

Another parameter used to characterize the amount of oil injected into the compressor is C_{ratio}. This parameter is defined by

$$C_{ratio} = \frac{\dot{m}_l c_l}{\dot{m}_g c_{p,g}} \tag{5.6}$$

which is the ratio of capacitance rates of the oil flow and the gas flow. Alternatively the oil mass fraction can be used, defined by

$$x_l = \frac{\dot{m}_l}{\dot{m}_l + \dot{m}_g} \tag{5.7}$$

where C_{ratio} and x_l are related through

$$x_l = \frac{C_{ratio}}{C_{ratio} + c_l/c_{p,g}} \tag{5.8}$$

which allows for conversion between the definitions of oil flooding amount.

5.2.1 Gas and Liquid Properties

For the Liquid-Flooded Ericsson Cycle experimental study upon which the model validation is based, the pressure ranged between 200 and 1500 kPa absolute, and temperature ranged between 300 K and 360 K. Over this envelope, the reduced temperature of the nitrogen (T/T_c) is near 3, and the reduced pressure (p/p_c) is between 0.05 and 0.42. Over this envelope, the compressibility factor Z ranges between 0.996 and 1.001 (Z=1.0 signifies a perfect gas), thus the nitrogen gas can be safely treated as being perfect. Thus the equation of state for the nitrogen gas is given by

$$\rho_g = \frac{p}{RT} \tag{5.9}$$

with the perfect gas assumption.

The perfect gas properties of nitrogen are found in Appendix C.1. Other refrigerants can be used with the scroll compressor model, and reference-accuracy residual Helmholtz formulation equations of state have been implemented for a wide range of refrigerants, as listed in Appendix C.2. The properties of the Zerol oil are found in Appendix C.4.

5.3 Mass Flow Models

In flooded scroll compressors, there are a number of different flow paths, encompassing the primary flow path from inlet to outlet, as well as well as the leakage flow paths. The physics of the flows in the primary flow path are significantly different than those of the leakages, and thus different models must be used for each category of flow. The selection of the flow models is presented here, and the description of the models follows.

As the flow enters into the compressor shell, it splits into two flow paths and these streams change direction to enter the suction pockets, as seen in Figure 5.1. Two models have been applied for this flow path, a flow through pipe bends model and a two-phase nozzle flow with a fictitious area correction coefficient. Both models were found to yield similar results, and the simpler fictitious area correction coefficient method was ultimately selected.

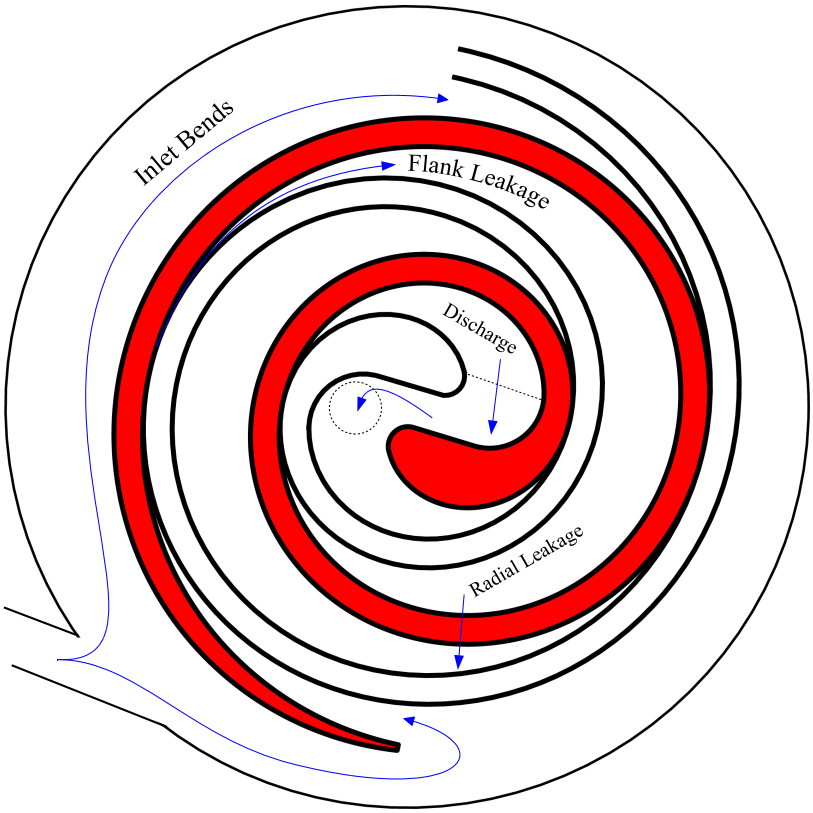


Figure 5.1. Flows in the scroll compressor.

After the flow of gas-liquid has entered into the suction chambers the next flow is from the d_1 and d_2 chambers to the dd chamber. As the geometry of this flow path is similar to that of a convergent nozzle, the flow is treated as that through a convergent nozzle with adiabatic, compressible two-phase flow. For the short time until pressure equilibrium occurs in the discharge region, the flow between the d_1 and d_2 chambers is very much like a converging nozzle.

For the leakage paths, the assumption that the flow path is frictionless is inaccurate, even in the case that the leakages are entirely filled with gas. This is due to the relatively long lengths of the leakage paths. For example, a typical radial leakage path might be 10 μm in height and 5 mm in length, which yields a length to height ratio of 500. Thus the leakage path is much more like a frictional flow between parallel plates than flow through an infinitely-thin nozzle. The leakage flows are taken to be entirely gas.

5.3.1 Two-Phase Compressible Flow In Nozzles

In the development of pressure drop in two-phase nozzles presented by Chisholm (1983), the derivation begins with conservation of linear momentum, with the mass flow between two control volumes expressed in the form

$$\dot{m} = C_d X_d A_{th} \sqrt{\frac{2\int_{p_{down}}^{p_{up}} v_e dp}{v_{e,down}^2 - \sigma v_{e,up}^2}} \tag{5.10}$$

where X_d is the area correction factor which has a value in the range 0 to 1. The discharge coefficient C_d is that given from the two-phase discharge coefficient model presented by Morris (1991), and its nominal value is 0.77. The application of the above mass flow model requires the area ratio σ of downstream to upstream areas, which does not have an analytical form for the scroll compressor geometry. The primary flow paths are approximated as having an area ratio (σ) of 0.0 since for each flow path, the upstream area is typically much greater than the throat area. Using the area ratio of zero yields the lowest flow rate possible with this model.

To apply the above mass flow model, the momentum effective specific volume (defined in the following section) of the mixture is integrated from the low pressure to the high pressure and then the mass flow rate is solved for as a function of known parameters. The integration is necessary because the momentum effective specific volume can vary with pressure since compressibility effects are taken into account. Thus the mass flow rate is then calculated from application of Eqn. (5.10)

Effective Mixture Properties

When a two-phase mixture of gas and liquid is flowing, and the two phases travel at different velocities, the effective density of the mixture is different from that when the mixture is at rest. The momentum effective specific volume is given based on a model from Chisholm (1983) which allows for entrainment of the liquid in the vapor phase which yields the effective mixture specific volume

$$v_e = [(1 - x_l)v_g + K_e x_l v_l] \left[(1 - x_l) + \frac{x_l}{K_e}\right] \tag{5.11}$$

where K_e is the so-called effective slip ratio, or the effective ratio of the speed of the gas to that of the liquid. The effective slip ratio can be obtained from

$$K_e = \left[\psi + \frac{(1 - \psi)^2}{K - \psi}\right] \tag{5.12}$$

where the entrainment slip ratio can be given by

$$K = \psi + (1 - \psi)\sqrt{\frac{1 + \psi x_l v_l/[(1 - x_l)v_g]}{1 + \psi x_l/(1 - x_l)}}\sqrt{\frac{v_g}{v_l}} \tag{5.13}$$

where Chisholm recommends a value of 0.4 for ψ. ψ is the mass fraction of the liquid that travels in the gas phase at the gas velocity. If ψ goes to unity, the slip ratio goes to unity and the mixture is treated as being homogeneous, and the effective specific volume goes to the homogeneous mixture specific volume. In the limit that ψ goes to zero, all the liquid travels in the liquid phase, and separated flow is obtained.

5.3.2 Radial And Flank Leakage

For the radial and flank leakage paths, the flow is both compressible and frictional, so in reality, the model employed should take both physical effects into account. One means of achieving this goal is to use the isentropic flow model, and derive a correction term to account for the frictional effects. The isentropic nozzle model is a simple model available in most fluid dynamics textbooks. The mass flow rate predicted by the isentropic nozzle flow model of pure gas is given as a function of the pressure ratio, given by

$$p_{ratio} = \begin{cases} \left(1 + \frac{(k-1)}{2}\right)^{k/(1-k)} & p_{down}/p_{up} <= \left(1 + \frac{(k-1)}{2}\right)^{k/(1-k)} \\ p_{down}/p_{up} & p_{down}/p_{up} > \left(1 + \frac{(k-1)}{2}\right)^{k/(1-k)} \end{cases} \tag{5.14}$$

where k is the ratio of specific heats, given by $k = c_p/c_v$. This allows for choking when the pressure ratio is less than the choking pressure ratio. Then the mass flow rate is given by

$$\dot{m}_{nozzle} = A\sqrt{p_{up}\rho_{up}}\sqrt{\frac{2k}{(k-1)}\left(p_{ratio}^{2/k} - p_{ratio}^{(k+1)/k}\right)} \tag{5.15}$$

when using the isentropic nozzle model.

The isentropic nozzle model does not take friction into account, so a correction term is derived in order to arrive at a frictionally-corrected flow rate. The results for the radial and flank gap correction term respectively are shown in Figures 5.2 and 5.3 respectively. The corrected mass flow rate can then be obtained from the equation

$$\dot{m} = \frac{\dot{m}_{nozzle}}{M} \tag{5.16}$$

where the correlation for the term M can be obtained from

$$M = \frac{a_0(L/L_0)^{a_1}}{a_2(\delta/\delta_0) + a_3}[\xi(a_4\mathrm{Re}^{a_5} + a_6) + (1-\xi)(a_7\mathrm{Re}^{a_8} + a_9)] + a_{10} \tag{5.17}$$

where the Reynolds number and the coefficients for each flow path can be found in Appendix C.5.

5.4 Forces

The forces of compression can be broadly categorized into thrust (or axial) force and radial force. The radial forces arrive as a result of the high-pressure gas acting normal to the walls of the scroll wraps, and the thrust loads as a result of the pressure difference between the shell pressure and the pressure in the compression chambers. The gas also exerts a force on the fixed scroll, but the primary focus of interest is the

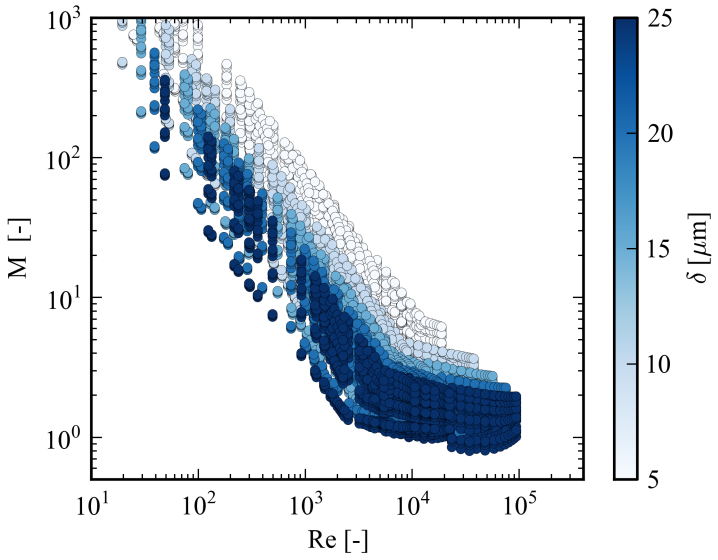


Figure 5.2. Ratio of prediction of mass flow rate from isentropic nozzle model and prediction of mass flow rate from full detailed flow model for the radial leakage.

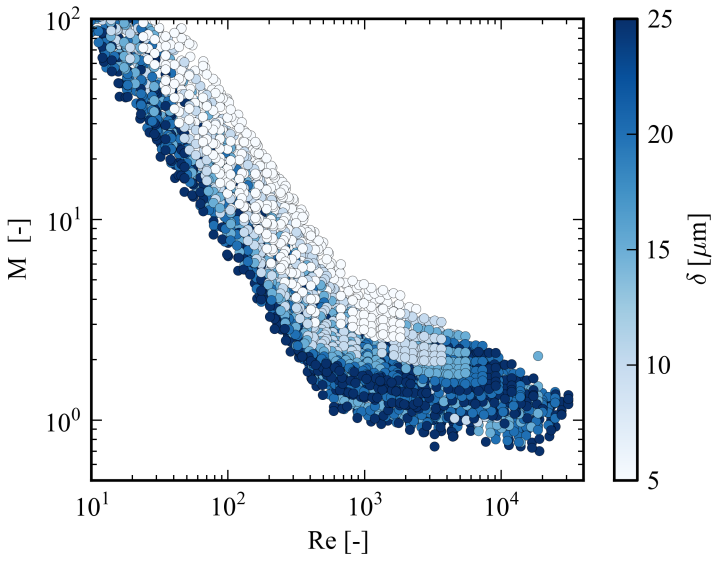


Figure 5.3. Ratio of prediction of mass flow rate from isentropic nozzle model and prediction of mass flow rate from full detailed flow model for the flank leakage.

force on the orbiting scroll. The forces on the fixed scroll are of primary interest with regards to material yielding from the stress load and the deflections of scroll wraps which may increase (or decrease depending on the leakage path under consideration) the flank leakages.

5.4.1 Radial Loads

The gas radial loads are calculated based on the geometric model presented in Chapter 4. Each of the radial force terms are expressed in the form $\mathbf{F}/p$, where p is the pressure of the gas in a chamber, and $\mathbf{F}$ is the force vector. As a result, the right-hand-side of each force term is then only a function of geometric parameters. Figure 5.4 shows the directions of the forces applied to the orbiting scroll.

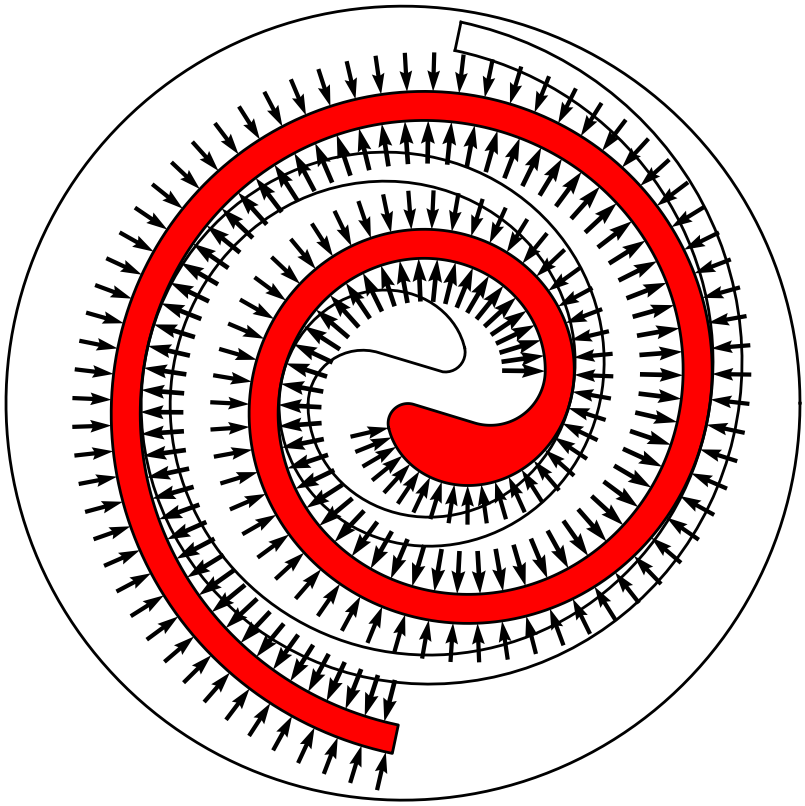


Figure 5.4. Pressure forces applied to the involute portion of the orbiting scroll.

The forces are expressed in this form to allow for decoupling of the geometric and thermodynamic models. Once the thermodynamic model has been run, the pressures in each chamber are then known, and the net force vector applied to the pin at the end of the shaft can be expressed as

$$\mathbf{F} = \sum_{CV=s1,s2...} \left(\frac{\mathbf{F}_{CV}}{p_{CV}} \right) p_{CV} \tag{5.18}$$

which sums the force contributions from each control volume. The instantaneous torque generated by the gas forces can be obtained by obtained by taking the cross product of a vector from the center of the shaft to the shaft pin and the net force vector. The vector from the center of the shaft to the pin location is given by

$$\mathbf{r} = r_o \cos(\phi_{ie} - \pi/2 - \theta)\hat{i} + r_o \sin(\phi_{ie} - \pi/2 - \theta)\hat{j} \tag{5.19}$$

and thus the instantaneous torque can be obtained from

$$\tau = \mathbf{r} \times \mathbf{F} \tag{5.20}$$

or

$$\tau = (\mathbf{r}_x \mathbf{F}_y - \mathbf{r}_y \mathbf{F}_x)\hat{k} \tag{5.21}$$

in component form. The average power generated to compress the gas is given by

$$\dot{W} = \overline{\tau}\omega \tag{5.22}$$

where $\overline{\tau}$ is the mean value of the torque over one rotation.

5.4.2 Axial Loads And Overturning Moment

A similar procedure is carried out to calculate the axial load generated by the compression chambers. The direction and magnitude of the force generated by the gas pressure will depend on the shell pressure which opposes the pressure in the chambers. This can be schematically seen in Figure 5.5. In the case shown here the

shell pressure is greater than that in the chambers and the net force tends to force the scroll set together. Thus the thrust load for each chamber is obtained from

$$\frac{\mathbf{F}_{axial}}{p_{CV} - p_{shell}} = \frac{V_{CV}}{h_s}\hat{\mathrm{k}} \tag{5.23}$$

which tends to push the scrolls apart. The forces from the scroll are unbalanced;

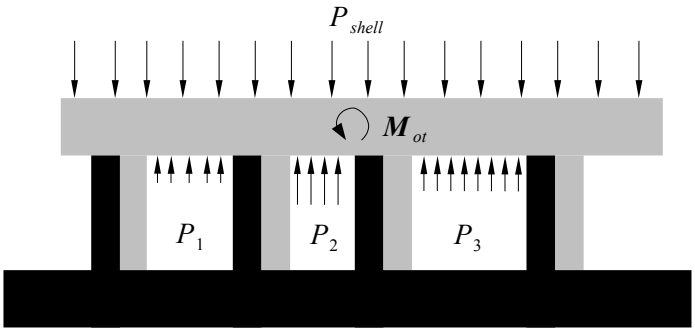


Figure 5.5. Schematic cross-section for calculation of axial load.

the centroid of the thrust load is not coincident with the axis passing through the center of the orbiting scroll. Thus each chamber also contributes to an overturning moment $\mathbf{M}_{ot}$ which tends to cause the orbiting scroll to come out of parallel with the fixed scroll, in the case shown in Figure 5.5, to rotate counter-clockwise. To calculate the total overturning moment, the overturning moment from each of the chambers is summed, and for one chamber, the overturning moment is given by

$$\mathbf{M}_{ot} = \mathbf{r}_{cent} \times \mathbf{F}_{axial} \tag{5.24}$$

where $\mathbf{r}_{cent}$ is the direction vector from the origin of the coordinate system for the orbiting scroll to the centroid of the chamber. The centroids for the chambers for a representative crank angle of $\pi/2$ radians are shown in Fig 5.6. The calculation of the centroids of the chambers was shown in Chapter 4.

5.5 Mechanical Losses

The vast majority of mechanical losses in scroll compressors arise from friction, through friction viscous losses, including those in the bearings and other contacting

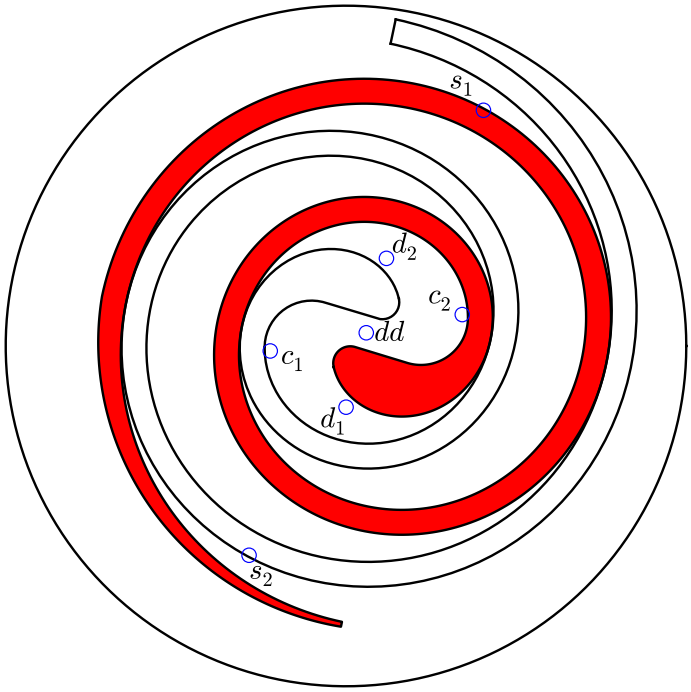


Figure 5.6. Centroids of Chambers at a crank angle of θ=$\pi/2$.

surfaces. There are two primary methods of modeling the impact of mechanical losses, which capture varying levels of the physical effects present in the system.

In the simplest way, the mechanical losses can be defined empirically as a function of the operating conditions. This is the method applied by Halm (1997), Chen (2002), Wang (2005), among many others. This method is also applied here, for which a constant mechanical loss torque is obtained, as proposed in Yanagisawa (1990). Further information on the model validation and tuning is available in Chapter 6.

A second method of calculating the mechanical losses is through the use of a full dynamic model for all the components. This is the method put forward by Ishii (1986), Chen (2002), and others. In order to carry out this method, the sum of forces and moments on each component is carried out such that the losses can be determined. This method requires a great deal more information about the mechanical interactions of the scroll compressor components.

5.6 Heat Transfer

Heat transfer in the scroll compressor occurs during a number of the processes that the refrigerant undergoes. In order to simplify the heat transfer analysis, the elements of the compressor are grouped into one lumped mass on which an energy balance is imposed. Figure 5.7 schematically shows the thermal interaction between the refrigerant and the lumped mass in the compressor. The analysis which follows allows for the prediction of the heat transfer rates of the inlet, compression, discharge, and exhaust processes. Various numbers of lumped masses can be used, but in the simplest case there is a single lumped mass that the heat transfer processes interact with.

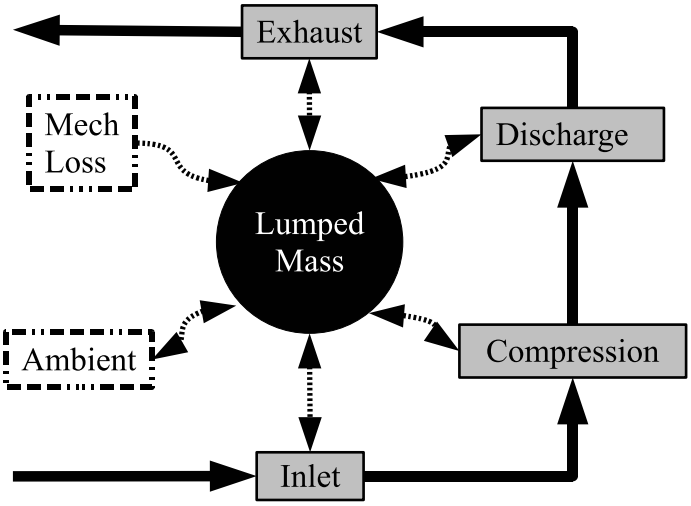


Figure 5.7. Map of heat flows in compressor.

5.6.1 Inlet/Exhaust Processes

As the refrigerant enters into the compressor shell, and exhausts out of the compressor shell, as shown in Figure 5.8, it exchanges heat with the body of the compressor, the lumped mass. The wall of the tubes are assumed to be at a fixed temperature,

that of the lumped mass. The outlet temperature of the path can be calculated from (Incropera and Dewitt, 2002)

$$T_o = T_{lump} - (T_{lump} - T_i) \exp\left(-\frac{\pi DL}{\dot{m} c_{p,m}} h_c\right) \tag{5.25}$$

based on the assumption of an isothermal boundary condition. The heat transfer coefficient h_c is obtained by assumption that the flow is turbulent and fully developed, in which case the Dittus-Boelter heat transfer coefficient is given by (Incropera and Dewitt, 2002)

$$h_c = 0.023 \frac{k_m}{D} \mathrm{Re}^{0.8} \mathrm{Pr}_m^{0.4} \tag{5.26}$$

and the Reynolds number is defined by

$$\mathrm{Re} = \frac{4\dot{m}}{\pi \mu_m D} \tag{5.27}$$

and thus the amount of heat transfer is then given by

$$\dot{Q}_{inlet,exhaust} = -\dot{m} c_{p,m} (T_o - T_i) \tag{5.28}$$

which is evaluated for both the inlet and outlet paths of the compressor.

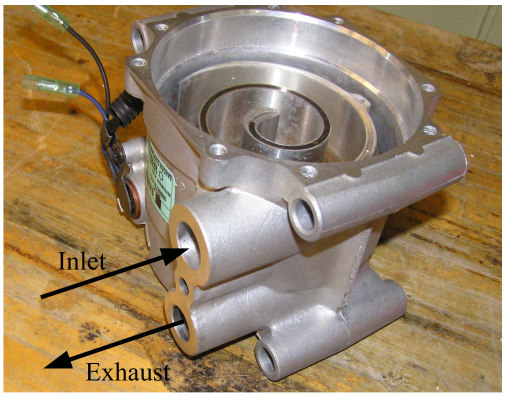


Figure 5.8. Inlet and outlet ports of compressor.

5.6.2 Scroll Heat Transfer

While the refrigerant is in the compression chambers it exchanges heat with the scroll wraps. Initially Halm (1997) carried out a simplified FEM analysis to show that the temperature of the scroll should be linear with the involute angle. Jang and Jeong (2006) experimentally measured the temperature in a scroll compressor under operation and their data validated Halm's assumption.

In order to calculate the heat transfer between the scroll wraps and the refrigerant in the chambers, the heat transfer coefficient is needed. In the case of the compression chamber, the heat transfer is given by a modified Dittus-Boelter term, with corrections for spiral geometry (Tagri and Jayaraman, 1962) as well as oscillation of the flow field(Jang and Jeong, 2006).

$$h_c = 0.023\left(1 + 1.77\frac{D_h}{r_c}\right)\left[1 + 8.48\left(1 - \exp(-5.35St)\right)\right]\frac{k_m}{D_h}\mathrm{Re}^{0.8}\mathrm{Pr}_m^{0.4} \tag{5.29}$$

where the Strouhal number, a non-dimensional frequency is given by

$$St = \frac{fA_{max}}{\overline{U}} \tag{5.30}$$

and the terms required in the Strouhal number expression are given by

$$\begin{aligned} f &= \frac{\omega}{2\pi} \\ A_{max} &= r_o \end{aligned} \tag{5.31}$$

where ω is the rotational speed of the compressor. The hydraulic diameter is given by

$$D_h = \frac{4r_o h_s}{2r_o + h_s} \tag{5.32}$$

which is that of a rectangular channel with width $2r_o$ and height h_s. The mean velocity $\overline{U}$ is obtained by taking half of the total mixture flow rate flowing through a channel with width $2r_o$ and height h_s (corresponds to the maximum flow area), for a mean velocity of

$$\overline{U} = \frac{\dot{m}_{total}}{4r_o h_s \overline{\rho}} \tag{5.33}$$

where $\overline{\rho}$ is the mean density, calculated at the mean of the suction and discharge temperatures and pressures. The Reynolds number is then given by

$$\mathrm{Re} = \frac{\overline{\rho}\overline{U}D_h}{\overline{\mu}} \tag{5.34}$$

where $\overline{\mu}$ is evaluated at the mean of the suction and discharge temperatures and pressures.

As described above, the temperature variation along the scroll wraps is taken as being linear with the involute angle, in which case the temperature along the scroll wrap is equal to

$$T(\phi) = T_{lump} + \left(\frac{dT}{d\phi}\right)(\phi - \phi_m) \tag{5.35}$$

where the mean involute angle is given by

$$\phi_m = \frac{\phi_{ie} + \phi_{is}}{2} \tag{5.36}$$

and the slope of the temperature- involute angle curve is given by

$$\frac{dT}{d\phi} = \frac{T_s - T_d}{\phi_{ie} - \phi_{is}} \tag{5.37}$$

where the involute angles are based on the inner involute, yielding a negative value for $dT/d\phi$. Thus the differential of heat transfer for a portion of one the scroll surfaces which form the control volume is given by

$$\delta\dot{Q}_{wall} = h_c h_s r_b(\phi - \phi_0)[T(\phi) - T_{CV}]d\phi \tag{5.38}$$

and carrying out the integration for the heat transfer for one scroll wall in one control volume yields

$$\dot{Q}_{wall} = h_c h_s r_b \left[\left(\frac{\phi^2}{2} - \phi_0\phi\right)(T_{lump} - T_{CV}) + \left(\frac{dT}{d\phi}\right)\left(\frac{\phi^3}{3} - \frac{(\phi_0 + \phi_m)\phi^2}{2} + \phi_0\phi_m\phi\right)\right]\Bigg|_{\phi_1}^{\phi_2} \tag{5.39}$$

where the angles ϕ_1 and ϕ_2 are the conjugate angles for each involute (inner and outer) which form the control volume. The calculations for the orbiting scroll involute and the fixed scroll involute are carried out separately. The heat transfer calculated for

each involute is added together to yield the total heat transfer rate for a given control volume.

The other surfaces which exchange heat with each control volume during operation are the top and bottom plates of each control volume. The heat transfer from the top and bottom plates is obtained by using an effective temperature of each plate. In practice, the temperature distribution in the metal of the plates is complex, governed by the balance of conduction through the metal of the plate and convective heat transfer to the gas. The average plate temperature $\overline{T}_{plate}$ is obtained as the average of the temperatures corresponding to the involute angles which bound the control volume. The area of each plate interacting with a control volume is equal to the volume of the chamber divided by the height of the scroll wraps. Thus the heat transfer from the top and bottom plates is equal to

$$\dot{Q}_{plates} = 2h_c \frac{V_{CV}}{h_s}(\overline{T}_{plate} - T_{CV}) \tag{5.40}$$

and the average heat transfer between the lump and the gas in the compression chambers is therefore obtained from

$$\overline{\dot{Q}_{scrolls}} = \frac{1}{2\pi}\int_0^{2\pi}\left[\sum_{CV}\left[\dot{Q}_{wall,in,CV} + \dot{Q}_{wall,out,CV} + \dot{Q}_{plates,CV}\right]\right]d\theta \tag{5.41}$$

which yields the average heat transfer rate over one revolution from a trapezoidal integration.

5.6.3 Ambient Heat Transfer

The heat transfer between the shell and the ambient is characterized by an overall thermal conductance value, in which case the amount of ambient heat transfer is given by

$$\dot{Q}_{amb} = UA_{amb}(T_{amb} - T_{lump}) \tag{5.42}$$

where T_{amb} is the ambient temperature.

5.6.4 Energy Balance

After all of the heat transfer terms have been calculated over one rotation, their average values are obtained, and the energy balance over the lumped mass is enforced, as described in Section 5.9.5. The energy balance is given by

$$\dot{Q}_{amb} + \dot{W}_{ML} + \dot{Q}_{inlet} - \overline{\dot{Q}_{scrolls}} + \dot{Q}_{exhaust} = r_{HT} \tag{5.43}$$

where r_{HT} is the residual, and a solver is used to drive the residual to zero, which enforces conservation of energy.

5.7 Conservation Equations

From the geometric model presented in Chapter 4, the laws of conservation of mass, energy, and liquid mass are expressed for each control volume. One typical control volume is seen in Fig. 5.9, where the relevant fluxes are shown.

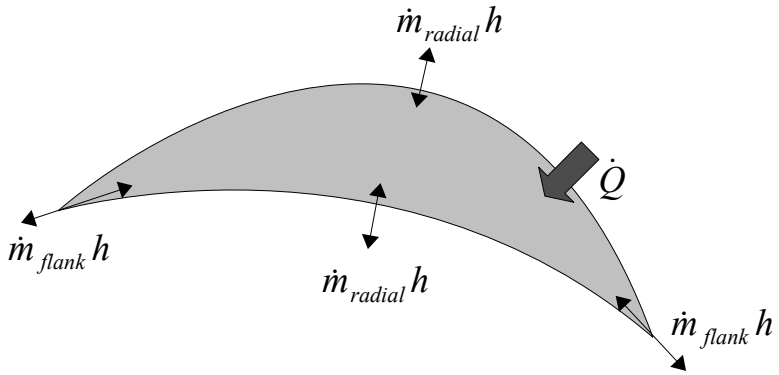


Figure 5.9. Schematic for Energy Fluxes for given control volume.

The derivatives of the properties of the fluid in the control volume are obtained from the conservation of energy, total mass, and oil mass. From this set of three conservation equations, the derivatives of three state variables can be obtained which are needed in further solver analysis. There are a number of sets of state variables that can be used, though since there are two components, three state variables are needed to fully fix the state of the mixture. Common pairs of state variables for dry

compression are temperature and density, temperature and pressure, or temperature and specific internal energy. The derivation using temperature, pressure and oil mass fraction as the state variables is presented in Appendix C.7, but the solution presented here, and that ultimately implemented, uses temperature, total mass and oil mass fraction as the state variables to fix the state. This set of variables minimizes the number of property derivatives that need to be evaluated numerically. Numerical differentiation of the property functions results in the generation of numerical noise and contributes to challenges of numerical stability and convergence.

5.7.1 Conservation Of Mass

The conservation of total mass for the control volume is given by

$$\frac{dm_{CV}}{dt} = \sum_f \dot{m} \tag{5.44}$$

where the sign of $\dot{m}$ is positive if the flow is into the control volume, and negative if the flow is out of the control volume. Ultimately it is more useful to express the derivatives with respect to the crank angle of the compressor, which requires the derivative transformation

$$\frac{d[\quad]}{dt} = \frac{d[\quad]}{d\theta}\frac{d\theta}{dt} \tag{5.45}$$

and since the rotational speed of the compressor is given by

$$\omega = \frac{d\theta}{dt} \tag{5.46}$$

and thus the conservation of mass for the control volume can be expressed as

$$\frac{dm_{CV}}{d\theta} = \frac{1}{\omega}\sum_f \dot{m} \tag{5.47}$$

in differential form.

5.7.2 Conservation Of Oil Mass

The conservation of oil mass follows a similar course to that of conservation of mass. The conservation of oil mass in the control volume is given by

$$\frac{dm_l}{d\theta} = \frac{1}{\omega}\sum_f \dot{m}x_{l,f} \tag{5.48}$$

where the term $x_{l,f}$ is the oil mass fraction of the flow path. The mass of liquid in the control volume is given by $m_l = x_l m_{CV}$, and the differential of the oil contained in the control volume by

$$dm_l = \frac{\partial m_l}{\partial m_{CV}}dm_{CV} + \frac{\partial m_l}{\partial x_l}dx_l \tag{5.49}$$

and evaluation of the partial derivatives and dividing through by $d\theta$ yields

$$\frac{dm_l}{d\theta} = x_l\frac{dm_{CV}}{d\theta} + m_{CV}\frac{dx_l}{d\theta} \tag{5.50}$$

and then substitution of the above result back into Eqn. (5.48) yields the conservation of oil mass of

$$\frac{dx_l}{d\theta} = \frac{1}{m_{CV}}\left[\frac{1}{\omega}\sum_f \dot{m}x_{l,f} - x_l\frac{dm_{CV}}{d\theta}\right] \tag{5.51}$$

in differential form.

5.7.3 Conservation Of Energy

To begin, the conservation of energy for the control volume is expressed as

$$\frac{dE}{dt} = m_{CV}\frac{du_{CV}}{dt} + u_{CV}\frac{dm_{CV}}{dt} \tag{5.52}$$

which assumes that the kinetic and potential energies of mass in the control volume is negligible, which is good assumption. Thus the specific energy in the control volume is simply equal to the specific internal energy of the fluid in the control volume. The specific internal energy of the fluid in the control volume (u_{CV}) can be expanded as a function of T, v and x_l, given by

$$du_{CV} = \frac{\partial u_{CV}}{\partial T}dT + \frac{\partial u_{CV}}{\partial v}dv + \frac{\partial u_{CV}}{\partial x_l}dx_l \tag{5.53}$$

which can be simplified by using the thermodynamic relations

$$c_v = \frac{\partial u_{CV}}{\partial T} \tag{5.54}$$

and

$$\frac{\partial u_{CV}}{\partial x_l} = u_l - u_g \tag{5.55}$$

and (Moran and Shapiro, 2008, p. 590)

$$\frac{\partial u_{CV}}{\partial v} = \left[T \left(\frac{\partial p}{\partial T} \right)_v - p \right] \tag{5.56}$$

which results in

$$du_{CV} = c_{v,m} dT + \left[T \left(\frac{\partial p}{\partial T} \right)_v - p \right] dv + (u_l - u_g) dx_l \tag{5.57}$$

and the overall conservation of energy can therefore be expressed as

$$\frac{dE}{dt} = \dot{Q} - p\frac{dV}{dt} + \sum_f \dot{m} h_f \tag{5.58}$$

where the term $-p\frac{dV}{dt}$ is the boundary work power term. The enthalpy h_f is that of the upstream state point of the flow path, so if the flow is out of the control volume, the enthalpy is that of the control volume, and if the flow is into the control volume, the enthalpy corresponds to that of the chamber that is flowing into the control volume. The summation of the product $\dot{m} h_f$ is taken over all flow paths interacting with the control volume. This yields the equation (after converting derivatives from time to crank angle)

$$m_{CV} \frac{du_{CV}}{d\theta} + u \frac{dm_{CV}}{d\theta} = \frac{\dot{Q}}{\omega} - p\frac{dV}{d\theta} + \frac{1}{\omega} \sum_f \dot{m} h_f \tag{5.59}$$

which is the expression of conservation of energy for the overall control volume. Expanding the left-hand-side of Eqn. (5.59) and employing the thermodynamic relation $u = h - pv$ yields

$$m_{CV} c_{v,m} \frac{dT}{d\theta} + m \left[T \left(\frac{\partial p}{\partial T} \right)_v - p \right] \frac{dv}{d\theta} + m_{CV} \frac{du}{dx_l} \frac{dx_l}{d\theta} + \frac{dm_{CV}}{d\theta} (h - pv) \tag{5.60}$$

where the derivative of the total volume ($V = m_{CV}v$) is given by

$$\frac{dV}{d\theta} = m_{CV}\frac{dv}{d\theta} + v\frac{dm_{CV}}{d\theta} \tag{5.61}$$

which can be substituted back into Eqn. (5.60). Substitution yields

$$m_{CV}c_{v,m}\frac{dT}{d\theta} + T\left(\frac{\partial p}{\partial T}\right)_v\left[\frac{dV}{d\theta} - v\frac{dm_{CV}}{d\theta}\right] + m_{CV}\frac{du}{dx_l}\frac{dx_l}{d\theta} + h\frac{dm_{CV}}{d\theta} - p\frac{dV}{d\theta} \tag{5.62}$$

and combining back the right-hand-side of Eqn. (5.59) yields

$$m_{CV}c_{v,m}\frac{dT}{d\theta} + T\left(\frac{\partial p}{\partial T}\right)_v\left[\frac{dV}{d\theta} - v\frac{dm_{CV}}{d\theta}\right] + m_{CV}\frac{du}{dx_l}\frac{dx_l}{d\theta} + h\frac{dm_{CV}}{d\theta} = \frac{\dot{Q}}{\omega} + \frac{1}{\omega}\sum_f \dot{m}h \tag{5.63}$$

which ultimately results in the derivative of the temperature with respect to the crank angle of

$$\frac{dT}{d\theta} = \frac{-T\left(\frac{\partial p}{\partial T}\right)_v\left[\frac{dV}{d\theta} - v\frac{dm_{CV}}{d\theta}\right] - m_{CV}(u_l - u_g)\frac{dx_l}{d\theta} - h\frac{dm_{CV}}{d\theta} + \frac{\dot{Q}}{\omega} + \frac{1}{\omega}\sum_f \dot{m}h}{m_{CV}c_{v,m}} \tag{5.64}$$

which is the conservation of energy in differential form.

The only parameter that cannot be directly calculated is $\left(\frac{\partial p}{\partial T}\right)_v$ which is obtained by a numerical derivative. To carry out the numerical derivative, a forward derivative is used while holding the specific volume constant. This can be expressed as

$$v_m(T, p, x_l) = v_m(T + \delta T, p + \delta p, x_l) \tag{5.65}$$

where the δp is determined for a given value of δT. The derivative term is then approximated by

$$\left(\frac{\partial p}{\partial T}\right)_v = \frac{\delta p}{\delta T} \tag{5.66}$$

for use in the derivatives above. If the oil mass fraction is zero (there is no flooding), the value of $\left(\frac{\partial p}{\partial T}\right)_v$ can be obtained directly from the gas's equation of state since the residual Helmholtz formulation gives the pressure as a function of temperature and density, or $p = f(T, \rho)$. If there is no oil flooding and the gas is treated as being ideal, $\left(\frac{\partial p}{\partial T}\right)_v = \rho R$

5.7.4 Summary

In summary, the derivatives of the properties of the fluid contained in a control volume can be expressed by the following set of three differential equations

$$\frac{dm_{CV}}{d\theta} = \frac{1}{\omega}\sum_{f}\dot{m} \tag{5.67}$$

$$\frac{dx_l}{d\theta} = \frac{1}{m_{CV}}\left[\frac{1}{\omega}\sum_{f}\dot{m}x_{l,f} - x_l\frac{dm_{CV}}{d\theta}\right] \tag{5.68}$$

$$\frac{dT}{d\theta} = \frac{-T\left(\frac{\partial p}{\partial T}\right)_v\left[\frac{dV}{d\theta} - v\frac{dm_{CV}}{d\theta}\right] - m_{CV}(u_l - u_g)\frac{dx_l}{d\theta} - h\frac{dm_{CV}}{d\theta} + \frac{\dot{Q}}{\omega} + \frac{1}{\omega}\sum_{f}\dot{m}h}{m_{CV}c_{v,m}} \tag{5.69}$$

and if there is no oil flooding, Eqn. (5.68) drops out, as does $\frac{dx_l}{d\theta}$ from Eqn. (5.69).

5.8 Solvers

As seen in Section 5.7, through the use of conservation of total mass, conservation of energy and conservation of oil mass, a system of differential equations for temperature, total mass and oil mass fraction in each of the control volumes can be obtained. One rotation of the crank of the compressor is divided into a number of small steps, where the index of a given step is i. The solver then steps through the crank angles, obtaining the properties of each control volume over an entire revolution.

The system of differential equations to be solved can be expressed as

$$\mathbf{f}_i = f(\mathbf{y}_i) = \begin{pmatrix} (dT/d\theta)_{i,1} \\ \vdots \\ (dT/d\theta)_{i,N_{CV}} \\ (dm_{CV}/d\theta)_{i,1} \\ \vdots \\ (dm_{CV}/d\theta)_{i,N_{CV}} \\ (dx_l/d\theta)_{i,1} \\ \vdots \\ (dx_l/d\theta)_{i,N_{CV}} \end{pmatrix} \tag{5.70}$$

where the subscript of the derivatives, ranging from 1 to the number of control volumes in existence at a given crank angle N_{CV}, is the index of the control volume. Thus the vector shown in Eqn. (5.70) contains all the derivatives of the properties for each control volume. Similarly, the vector $\mathbf{y}_i$ given by

$$\mathbf{y}_i = \begin{pmatrix} (T)_{i,1} \\ \vdots \\ (T)_{i,N_{CV}} \\ (m_{CV})_{i,1} \\ \vdots \\ (m_{CV})_{i,N_{CV}} \\ (x_l)_{i,1} \\ \vdots \\ (x_l)_{i,N_{CV}} \end{pmatrix} \tag{5.71}$$

contains the state variables for all the control volumes at the step i.

5.8.1 Euler Method

Of the available solvers for integration of systems of ordinary differential equations, the Euler method is by far the easiest method to implement, though it does have

weaknesses. To implement this method, the derivatives of properties are evaluated at the i-*th* step, and then the derivatives from the i-*th* step are used to calculate the property values at the i+1-*th* step though the application of

$$\mathbf{y}_{i+1} = \mathbf{y}_i + \Delta\theta \cdot \mathbf{f}_i \tag{5.72}$$

at each step for each control volume. This method is simple to implement; the only computational work that is required is to evaluate Eqn. (5.70) at the i-th step, and with known values for $\mathbf{y}_i$, the values for $\mathbf{y}_{i+1}$ can be readily evaluated.

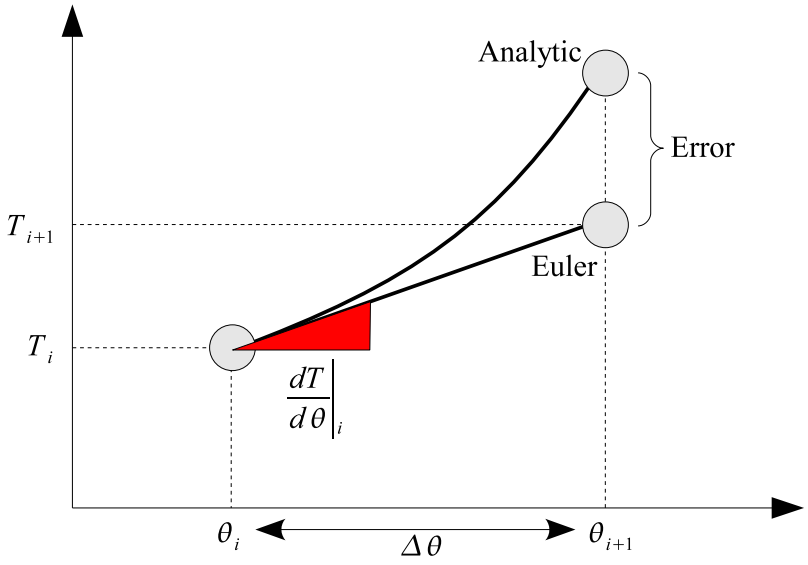


Figure 5.10. Schematic of one step of the simple Euler method.

Figure 5.10 shows a schematic view of the application of the simple Euler method to the derivative of temperature with respect to the crank angle. The simple Euler method linearly extrapolates the temperature based on the derivative at the i-*th* step to the i+1-*th* step. For a large step, the error between the analytic solution and the approximate solution can be large. To achieve higher accuracy and approach the analytic solution, the step size must be made smaller. The same basic principles hold for all the more advanced ODE solvers presented below.

The primary disadvantage to the Euler method for application to the modeling of scroll compressors is that the Euler method has a small range of stable operation.

Practically, stability can be nearly assured using the Euler method if the step size is made small enough. The disadvantage of needing many steps per rotation is that much more computational work is required per crank rotation. In general, as the number of equations in a system of ODEs increases, the system is more and more likely to experience numerical stiffness.

A second disadvantage to the Euler method is that it is only a first-order truncation error scheme. This means that the error from the truncation of the infinite series used to generate Eqn. (5.72) is proportional to the step size used. To halve the truncation error, the step size must be halved. As a result, quite small steps must be used to minimize the truncation error and obtain accurate results over one rotation.

5.8.2 Backward Euler Method

If the Euler method cannot be used due to problems with stability, the next candidate would be an implicit method, such as the Backward Euler method. In the Backward Euler method, the derivative vector $\mathbf{f}$ is evaluated at the $i+1$-th step rather than at the i-th step. Thus the solution method is given by the application of

$$\mathbf{y}_{i+1} = \mathbf{y}_i + \Delta\theta \cdot \mathbf{f}_{i+1} \tag{5.73}$$

at each step for each control volume. Practically this means that to obtain the values of temperature, pressure and oil mass fraction at the $i+1$-th step, guesses must be made for their values at the $i+1$-th step, and an iterative nonlinear system solver, like Newton-Raphson is then used to solve for $\mathbf{y}_{i+1}$. This procedure is very computationally expensive because each iteration of the Newton-Raphson solver requires an evaluation of the Jacobian matrix.

To use the Newton-Raphson method to solve for the properties at the $i+1$-th step, the first step is the calculation of the residual vector using the first guess of the new values. In this case, the residual vector is given by

$$r(\mathbf{y}_{i+1}) = \mathbf{y}_{i+1} - \mathbf{y}_i - \Delta\theta \cdot f\left(\mathbf{y}_{i+1}\right) \tag{5.74}$$

and the Jacobian matrix is composed by

$$\mathbf{J} = \begin{bmatrix} | & & | \\ J_1 & \dots & J_{3N} \\ | & & | \end{bmatrix} \tag{5.75}$$

and thus the j-th column of the Jacobian matrix (evaluated by the use of forward differences) is given by

$$J_j = \frac{r(y_{i+1,1}, y_{i+1,2}, ...y_{i+1,j} + \Delta y_j, y_{i+1,j+1}, y_{i+1,j+2}, ...) - r(\mathbf{y}_{i+1})}{\Delta y_j} \tag{5.76}$$

where each column is formed by incrementing the corresponding value in the variable vector in order to build the numerical derivative. The next guess for the values of $\mathbf{y}_{i+1}$ are obtained by then solving

$$\mathbf{y}_{i+1} = \mathbf{y}_{i+1} - r\left(\mathbf{y}_{i+1}\right)\left(\mathbf{J}_{i+1}\right)^{-1} \tag{5.77}$$

and Eqns. (5.74) to (5.77) are then repetitively applied until subsequent evaluations result in a change less than some convergence criterion ε. The use of this method is not recommended, but is provided for completeness, and may be necessary for very stiff systems of equations. Even so, the adaptive RK4/5 method presented below still appears to perform better, even though it is not an unconditionally stable method like the Backward Euler method.

5.8.3 Semi-Implicit Backward Euler

From the analysis presented above on the Backward Euler method, it should be evident that both the Euler and Backward Euler methods have disadvantages. One means of achieving improved stability and relatively efficient computation is to use the Semi-Implicit Backward Euler method. This solver is also known by the name Semi-Explicit Backward Euler. The semi-implicit nature of this solver is due to the treatment of the derivatives.

While in the Euler method the derivatives are evaluated at the i-th step, and in the Backward Euler method the derivatives are evaluated at the $i+1$-th step, for the

Semi-Implicit Backward Euler method, the derivatives are evaluated at the $i+1$-th step, but are linearized and then approximated based on known values from the i-th step.

To arrive at this formulation, first the derivative of the Backward Euler method is linearized, yielding

$$\mathbf{y}_{i+1} = \mathbf{y}_i + \Delta\theta \cdot \left(\mathbf{f}_i + \frac{d\mathbf{f}_i}{d\mathbf{y}_i} \left(\mathbf{y}_{i+1} - \mathbf{y}_i \right) \right) \tag{5.78}$$

and thus all the terms in Eqn. (5.78) are explicit, and the value at the $i+1$-th step can be obtained from

$$\mathbf{y}_{i+1} = \mathbf{y}_i + \Delta\theta \cdot \left(1 - \Delta\theta \frac{d\mathbf{f}_i}{d\mathbf{y}_i} \right)^{-1} \mathbf{f}_i \tag{5.79}$$

which is an explicit formulation. In this case the Jacobian matrix $\dfrac{d\mathbf{f}_i}{d\mathbf{y}_i}$ is obtained by the method of Eqn. (5.76), but the "residual" vector is equal to $\mathbf{y}$. While the stability characteristics of the SIBE method are an improvement over the simple Euler method, a large amount of computational work is required per step to generate the Jacobian, and it is only a first order truncation error method. All in all, the SIBE method is an improvement over the forward Euler method, but does not offer the excellent performance of the adaptive Runge-Kutta solver presented below.

5.8.4 RKF Adaptive Runge-Kutta Solver

One of the most popular solvers for systems of ordinary differential equations is the Runge-Kutta 4^{th} order solver (RK4). This solver operates by taking a few partial steps over the course of a single step, and then uses an averaging method to obtain the output value vector at the next step. The RK4 method is $4th$ order, so the halving of $\Delta\theta$ should divide the error per step by a factor of 2^4 or 16. In practice, other errors will contribute to the total error such that the error is not quite proportional to the step size to the $4th$ power. The RK4 method is used extensively in commercial ODE solver codes. It is very well suited to non-stiff systems of equations. The disadvantage

of the the standard RK4 method is that the step size is fixed, and this means that sometimes unnecessarily small steps are being taken, and other times, the solver's solution process can go unstable if the step size is too large. RK4 has similar stability behavior to the simple Euler method.

The middle way then is an adaptive Runge-Kutta method where the step size is dynamically selected in order to maintain the maximum truncation error per step ε_{max} below some desired level $\varepsilon_{allowed}$. The adaptive method is numerically efficient and offers very good stability. To carry out the adaptive method, 6 different derivatives are calculated, obtained from the following set of equations for a step size of $\Delta\theta$ (Kreyszig, 2006, p. 894)

$$\begin{aligned}
\mathbf{k}_1 &= \Delta\theta f\left(\theta_i, \mathbf{y}_i\right) \\
\mathbf{k}_2 &= \Delta\theta f\left(\theta_i + \tfrac{1}{4}\Delta\theta, \mathbf{y}_i + \tfrac{1}{4}\mathbf{k}_1\right) \\
\mathbf{k_3} &= \Delta\theta f\left(\theta_i + \tfrac{3}{8}\Delta\theta, \mathbf{y}_i + \tfrac{3}{32}\mathbf{k}_1 + \tfrac{9}{32}\mathbf{k}_2\right) \\
\mathbf{k}_4 &= \Delta\theta f\left(\theta_i + \tfrac{12}{13}\Delta\theta, \mathbf{y}_i + \tfrac{1932}{2197}\mathbf{k}_1 - \tfrac{7200}{2197}\mathbf{k}_2 + \tfrac{7296}{2197}\mathbf{k}_3\right) \\
\mathbf{k}_5 &= \Delta\theta f\left(\theta_i + \Delta\theta, \mathbf{y}_i + \tfrac{439}{216}\mathbf{k}_1 - 8\mathbf{k}_2 + \tfrac{3680}{513}\mathbf{k}_3 - \tfrac{845}{4104}\mathbf{k}_4\right) \\
\mathbf{k}_6 &= \Delta\theta f\left(\theta_i + \tfrac{1}{2}\Delta\theta, \mathbf{y}_i - \tfrac{8}{27}\mathbf{k}_1 + 2\mathbf{k}_2 - \tfrac{3544}{2565}\mathbf{k}_3 + \tfrac{1859}{4104}\mathbf{k}_4 - \tfrac{11}{40}\mathbf{k}_5\right)
\end{aligned} \tag{5.80}$$

which yield 4^{th} and 5^{th} order steps and the difference of 4^{th} and 5^{th} order steps based on the combination of the $\mathbf{k}$ coefficients. The 5^{th} order approximation of the next step is therefore given by

$$\mathbf{y}_{i+1} = \mathbf{y}_i + \frac{16}{135}\mathbf{k}_1 + \frac{6656}{12825}\mathbf{k}_3 + \frac{28561}{56430}\mathbf{k}_4 - \frac{9}{50}\mathbf{k}_5 - \frac{2}{55}\mathbf{k}_6 \tag{5.81}$$

and an approximation of the error vector is given by

$$\epsilon = \Delta\theta\left(\frac{1}{360}\mathbf{k}_1 - \frac{128}{4275}\mathbf{k}_3 - \frac{2197}{75240}\mathbf{k}_4 + \frac{1}{50}\mathbf{k}_5 + \frac{2}{55}\mathbf{k}_6\right) \tag{5.82}$$

which yields approximations for the errors for the given value of $\Delta\theta$. This error vector is only an approximation since there are higher order truncation terms that are neglected, but the error vector adequately predicts the error per step. Therefore it is possible to determine the worst error, which can be given by

$$\varepsilon_{max} = max(|\epsilon_i|) \text{ for } i = 1...3N \tag{5.83}$$

where the function $max()$ yields the maximum value. Some authors have proposed to normalize the error vector terms if they are of greatly different orders of magnitude, but for the scroll compressor modeling, error normalization does not seem to be necessary.

If error is below tolerance ($\varepsilon_{allowed}$), the new step size for the next step is given by

$$\Delta\theta_{new} = 0.9\Delta\theta_{old}\left(\frac{\varepsilon_{allowed}}{\varepsilon_{max}}\right)^{0.2} \tag{5.84}$$

which will increase the step size since $\varepsilon_{allowed} > \varepsilon_{max}$. If the error is above tolerance (ε_{max}), the new step size for the failed step is given by

$$\Delta\theta_{new} = 0.9\Delta\theta_{old}\left(\frac{\varepsilon_{allowed}}{\varepsilon_{max}}\right)^{0.3} \tag{5.85}$$

which will decrease the step size since $\varepsilon_{allowed} < \varepsilon_{max}$. The step is retried, and is continually retried until $\varepsilon_{max} < \varepsilon_{allowed}$.

If at each step of the revolution the step size is selected by an idealized step-size algorithm which yields exactly the imposed maximum error per step, 6 evaluations of the property derivative function f() per step are required over the course of the revolution. Due to the fact that the stiffness of the system of equations is constantly changing, the real adaptive method must also change the step size to keep the error right at the maximum allowable error. The use of too small a step result in unnecessary computational work, while the use of too large a step results in an unacceptably large error. For one run of the model, it was found that an average of 6.13 evaluations of the derivative function per step were required, which suggests that the step up-and-down-sizing relations in Eqns. (5.84) and (5.85) work very well at keeping the step size near the ideal step size throughout the compression process, minimizing the amount of computational work for a given imposed accuracy. The results in Section 5.10 show the variation in step size over the course of a rotation.

Selection of the maximum allowable error per step $\varepsilon_{allowed}$ requires some finesse. If it is made too small, the model will work too hard for each rotation. If it is too large, the overall error per rotation is too large, and the outer loop that enforces continuity

of the rotations may not converge. A value for $\varepsilon_{allowed}$ on the order of $1\text{x}10^{-6}$ seems to work well. On balance, it is better to err on the side of too small a step size, while still staying at least a few orders of magnitude above machine precision.

5.8.5 Numerical Considerations

One of the fundamental characteristics of compressor models in general is that they often have many solvers nested inside each other like Russian stacking dolls. In general, in the process of running the model, the innermost model solves, and then passes its output to the next most outermost solver. The selection of convergence criterion for each solver in the stack of solvers is of utmost importance as a simple example here will demonstrate.

Suppose that we have a simple function like

$$y = ax^2 - b \tag{5.86}$$

and we want to find the value of x that yields $y(x) = 0$ and then take the derivative of the solution with respect to the coefficient a. This process is similar to what happens in the property code to calculate the enthalpy for a given set of pressure and entropy using a secant method embedded in another secant method. To carry out this analysis numerically, the function in Equation 5.86 is solved for x using a secant method for a given value of a until the convergence criterion

$$|y(a, x)| < \varepsilon \tag{5.87}$$

is satisfied. Then the numerical derivative of x with respect to a is calculated using a first-order truncation error forward difference, as in

$$\frac{dx}{da} = \frac{x(a + \Delta a) - x(a)}{\Delta a} \tag{5.88}$$

The analytic solution to the problem proposed in this section is straightforward, and is given by

$$x = \sqrt{\frac{b}{a}} \tag{5.89}$$

$$\frac{dx}{da} = -\frac{\sqrt{b}}{2a^{3/2}} \tag{5.90}$$

which allows for the comparison between the numerical and analytic solutions to investigate the challenges of the numerical solution. With the parameters a=0.3, b=0.5, the analytic solution for the value of dx/da is -2.151. Figure 5.11 shows the results of the nested solving; each of the steps in the derivative corresponds to one further step in the inner loop solver. As the convergence criterion for the inner loop is decreased, the error of the derivative is reduced.

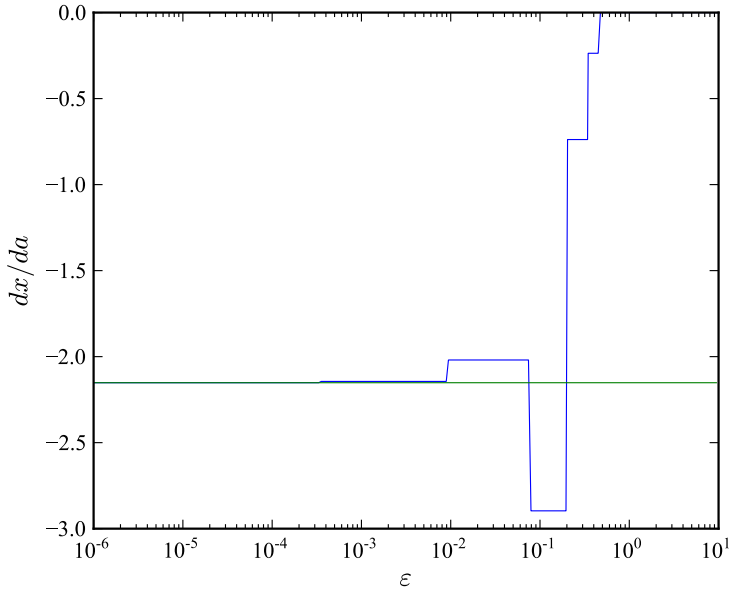


Figure 5.11. Derivative of root as function of inner loop convergence criterion.

5.9 Model Closure

Calculating all the state variables for all the chambers over the course of a rotation involves a number of solvers nested inside each other. Before these solvers can begin, the model must be initialized with starting values and then the solving process can begin. During the compression process, a number of events happen, including merging and splitting of discharge chambers.

5.9.1 Model Initialization

Before the model is run, the mass flow rate is not known, and the mass flow rate is needed in order to calculate inlet heat transfer. Therefore for the first time through the model, the total flow rate is set to be equal to the the flow rate that would be achieved with a volumetric efficiency of 100%. The inlet heat transfer can then be calculated, which yields a prediction for the mixture temperature entering into the suction area. In subsequent iterations of the model, the prediction of the mixture flow rate is refined, and the prediction for the inlet heat transfer rate is improved.

5.9.2 Chamber Initialization

At the beginning of the model execution before a rotation has been attempted, no values are available for the properties of the mixture in the control volumes. For the suction chambers s_1 and s_2 and the suction channel sa, the initial guess value is that the pressure and liquid mass fraction are equal to the suction properties upstream of the compressor. The temperature of the fluid in these chambers is set equal to the value after having gone through the inlet heat transfer process.

The initial guess value for the properties of the fluid in the discharge chamber ddd is that it is at the discharge temperature, pressure and oil mass fraction. The discharge pressure is an input to the model, and the discharge oil mass fraction must be equal to that of the suction oil mass fraction due to conservation of mass. In

order to calculate the initial discharge temperature estimation, an assumption about the compression process must be made. The ratio of specific heats k^*, defined by Hugenroth (2006) as

$$k^* = \frac{x_l c_l + (1 - x_l)\, c_{p,g}}{x_l c_l + (1 - x_l)\, c_{v,g}} \tag{5.91}$$

is used, which assumes that the overall compression of the two-phase mixture can be treated as that of an adiabatic compression of a quasi-perfect gas. Therefore the first guess value of the discharge temperature T_d can be obtained from

$$\begin{aligned} T_{d,isen} &= T_s \left(\frac{p_d}{p_s}\right)^{\frac{k^* - 1}{k^*}} \\ T_{disc} &= \frac{(T_{d,isen} - T_s)}{\eta_{guess}} + T_s \end{aligned} \tag{5.92}$$

where η_{guess} is a guess for the overall isentropic efficiency of the compressor. Many, if not all, of the assumptions used to get this predicted discharge temperature are invalid. Fortunately the analysis presented here is only used to arrive at a predicted discharge temperature, and the inaccuracy in the prediction will only cause the model to take longer to converge to the final discharge temperature.

For the outermost compression chamber with index α=1, which is what the suction chamber turns into at the end of the first rotation, it is assumed that the fluid in the suction chamber comes back to near the suction state, thus the outermost compression chamber is initialized with the suction properties. Depending on the design of the scroll wraps it is possible that there are multiple pairs of compression chambers, and if so, the compression chambers are initialized with temperature and pressure values based on an adiabatic, isentropic compression. The volume ratio for each compression chamber over one revolution is known based on the geometry, and the initial pressure

and temperature of the next pair of chambers towards the center of the compressor can be obtained from

$$\begin{aligned} p_{c,\alpha+1} &= p_{c,\alpha}\left[\frac{V_c(\alpha,\theta=2\pi)}{V_c(\alpha,\theta=0)}\right]^{k^*} \\ T_{c,\alpha+1} &= T_{c,\alpha}\left[\frac{V_c(\alpha,\theta=2\pi)}{V_c(\alpha,\theta=0)}\right]^{k^*-1} \end{aligned} \tag{5.93}$$

where α is equal to one for the compression chamber attached to the suction chamber. There are N_c pairs of compression chambers, given from Eqn. (4.108)

5.9.3 Revolution

In order to carry out one revolution, the state points are initialized based on the procedure laid out in the previous section. Then to determine the temperature, pressure and oil mass fraction in each chamber over the revolution, three procedures are carried out at each step of the revolution:

- Mass flow between all the chambers is determined
- Heat transfer between the scrolls and the gas/liquid mixture is calculated
- The ODE system solver is used to predict the property values at the next step

This procedure is carried out for all the points over the course of the rotation. There are two critical points in the compression process that require special handling. These are the compression angles at which the discharge process begins, and the angle at which the d_1, d_2, and dd chambers have all equalized in pressure. The discharge angle is known from the geometric model but the mixing angle is dependent on the compression process.

The Discharge Angle

When the adaptive Runge-Kutta solver is used, it cannot handle a step that traverses the discharge angle since some of the chambers that are defined right before

the discharge chamber are not defined right after the discharge angle. In particular, the innermost set of compression chambers gets swallowed into the discharge region, and the residual volume in the merged *ddd* chamber becomes the *dd* chamber. As a result, it is necessary to redefine the chambers around the discharge angle. In practice if the given step size would end up beyond the discharge angle, a small step is taken with the adaptive solver turned off to bring the crank angle just short of the discharge angle ($\theta = \theta_d - 1.0 \text{ x } 10^{-10}$ works well). After redefining the chambers, another small step of $\Delta\theta = 2.2 \text{ x } 10^{-10}$ radians is taken to bring the solver just into the discharge region. This size of step is taken to avoid the crank angle θ_d (at which point some volumes are not defined) because the adaptive RK solver takes a few mini-steps, one of which is at $\Delta\theta/2$.

For the step that traverses the discharge angle, the volumes, temperatures, pressures, etc. corresponding to the newly defined chambers are given by the chamber that brought it into being. The innermost pair of compression chambers c_{1,N_c} and c_{2,N_c} become the discharge chambers d_1 and d_2 respectively since they are now open to the discharge region. The *dd* chamber is now defined to be what is left over in the discharge chamber *ddd*. In mathematical terms, this means that for this step, and only for this step,

$$\begin{array}{lll} V_{d1} = V_{c1,N_c} & V_{d2} = V_{c2,N_c} & V_{dd} = V_{ddd} \\ T_{d1} = T_{c1,N_c} & T_{d2} = T_{c2,N_c} & T_{dd} = T_{ddd} \\ p_{d1} = p_{c1,N_c} & p_{d2} = p_{c2,N_c} & p_{dd} = p_{ddd} \\ \vdots & \vdots & \vdots \end{array} \tag{5.94}$$

which results in a large amount of bookkeeping to ensure that all the properties and other parameters are handled properly at this point. After the discharge angle, the chambers d_1, d_2, and *dd* properly exist and are treated as normal control volumes.

The Merging Process

Once the discharge angle has been passed, the d_1, d_2, and *dd* chambers begin to equalize in pressure, and they are then mixed when the difference in pressure is less

than some convergence criterion. When the innermost compression chamber pressure is lower than the discharge chamber pressure at the discharge angle, the merging process tends to proceed smoothly, and in usually in less than a quarter rotation the pressures of the chambers have equalized to within 0.02%. The mass flow rates and the derivatives of the chamber volumes with respect to the crank angle both tend to drive the chamber pressures together. Numerically the system of equations becomes more stiff as the flow area between discharge chambers increases relative to the volume of the *dd* chamber, but as long as the adaptive-RK solver is being used, this just means that the step size will decrease slightly to avoid instability. Typically there are no major numerical problems in getting the chambers to merge in the under-compression case.

When the gas is over-compressed in the compression process and arrives at the discharge angle above the discharge pressure, achieving pressure equilibrium is more problematic. While the flow area between chambers in the discharge region is relatively large, the derivatives of volume tend to drive the pressures of the chambers apart. This can be understood by considering Figure 5.12, which shows the derivatives of the volumes in the discharge region. Since now the derivative of volume of d_1 chamber is negative (and the volume of the d_1 chamber is much larger than the *dd* chamber - see Figure 4.34(a)), the derivative tends to want to increase the pressure of the d_1 chamber, driving the pressures of the d_1 and *dd* chambers apart. In contrast, the negative derivative of the volume of the d_1 chamber in the under-compressed case tends to want to drive the chamber pressures together. A more relaxed convergence criterion must be used for the merging, otherwise the chambers will never merge. A convergence criterion of 1% is used, and generally enables merging, but this causes problems in getting the solver for rotation convergence to converge. The numerical handling of merging with over-compression could certainly be improved, though there do not seem to be any obvious avenues for progress.

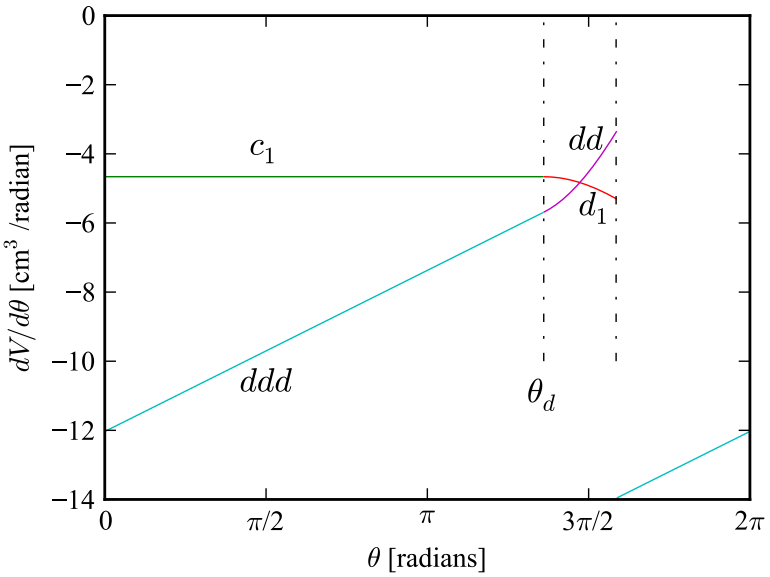


Figure 5.12. Derivative of volumes with respect to crank angle for discharge chambers for the Sanden compressor.

Once the merging pressure has been reached, balance equations are used to determine the state for the new merged chamber *ddd*. The volume of the merged chamber is equal to

$$V_{ddd} = V_{dd} + V_{d1} + V_{d2} \tag{5.95}$$

which simply sums the contributions from all the constituent volumes. Since at the merging angle, the pressures of all the chambers are by definition close to each other, a volume-weighted average pressure can be used for the new chamber, given by

$$p_{ddd} = \frac{V_{dd}p_{dd} + V_{d1}p_{d1} + V_{d2}p_{d2}}{V_{ddd}} \tag{5.96}$$

and the liquid mass fraction for the merged chamber is given by

$$x_{l,ddd} = \frac{x_{l,d1}m_{d1} + x_{l,d2}m_{d2} + x_{l,dd}m_{dd}}{m_{d1} + m_{d2} + m_{dd}} \tag{5.97}$$

which is a mass-weighted average of all the chambers. The temperatures of the chambers are in general not equivalent because there is no driving potential that tends to drive them together, thus an energy balance must be used to find the mixture

temperature. If the mixing process is assumed to be adiabatic, the total internal energy must be conserved. This means that the specific internal energy of the new chamber can be obtained from

$$u_{ddd} = \frac{u_{d1}m_{d1} + u_{d2}m_{d2} + u_{dd}m_{dd}}{m_{d1} + m_{d2} + m_{dd}} \tag{5.98}$$

and the temperature of the merged chamber T_{ddd} can be obtained iteratively by solving

$$u_{ddd} = u_m(T_{ddd}, p_{ddd}, x_{l,ddd}) \tag{5.99}$$

since u_{ddd}, p_{ddd} and $x_{l,ddd}$ are all now known.

The *ddd* chamber proceeds to the end of the rotation, and in fact continues on to the next time the discharge angle is reached, which could be the same rotation or the following rotation.

5.9.4 Wrapping

At the end of the revolution the initial values for the next revolution are updated by applying the ending values from the current revolution. This means that at the end of the rotation, the temperature and pressure of the following chambers are wrapped based on the redefinitions listed in Table 5.1. The wrapping process is considered to be completed when the values at the end of the rotation are equal to those at the beginning of the rotation to within some tolerance. This means that for the rotation to be converged, the following inequality must hold:

$$|\chi_{old} - \chi_{new}| < \varepsilon_{wrap} \tag{5.100}$$

where the inequality must hold for all the control volumes and χ is in the set of all the state variables - T, p, ρ, etc. If the inequality does not hold, the values of χ_{new} at θ=0 are set to be the values of χ_{old} at θ=2π. In addition, the values for s_1 and s_2 are reinitialized to their values from the beginning of the rotation.

The first three rotations of one run of the model are shown in Figure 5.13. This shows that the initial guesses were quite good, with the exception of the discharge

Table 5.1 Wrapping chamber definitions.

Old Chamber		New Chamber
sa	$\rightarrow$	sa
s_1	$\rightarrow$	$c_{1,\alpha}$
s_2	$\rightarrow$	$c_{1,\alpha}$
$c_{1,\alpha}$	$\rightarrow$	$c_{1,\alpha+1}$
$c_{2,\alpha}$	$\rightarrow$	$c_{2,\alpha+1}$
$\vdots$		$\vdots$
c_{1,N_c-1}	$\rightarrow$	c_{1,N_c}
c_{2,N_c-1}	$\rightarrow$	c_{2,N_c}
ddd	$\rightarrow$	ddd

pressure which is higher due to the pressure drop through the discharge port. Beyond the first rotation it is difficult to visually see the rotation convergence process proceed.

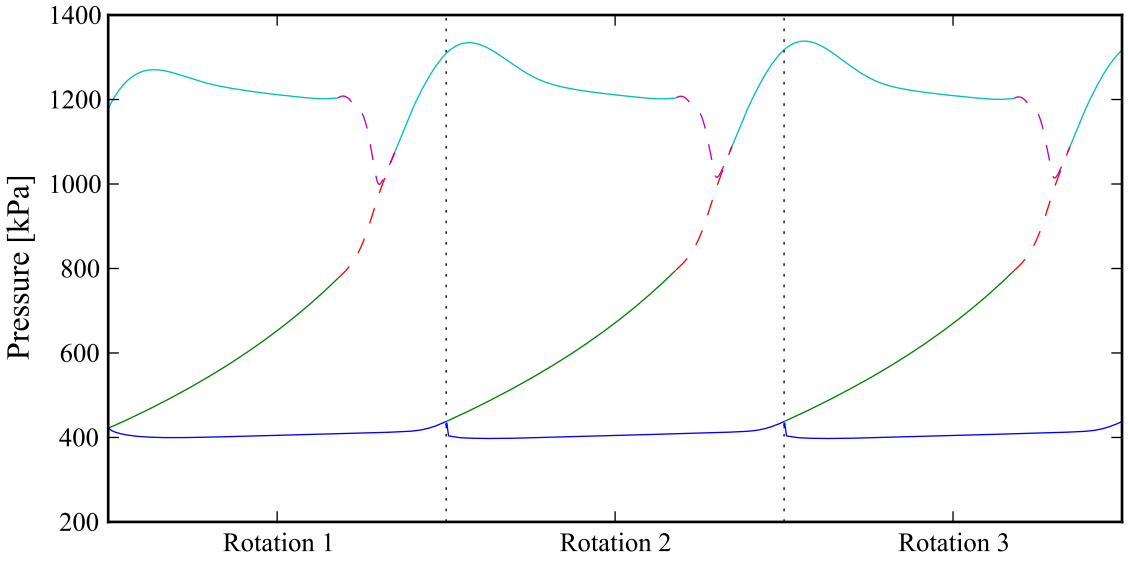


Figure 5.13. First three rotations of the compressor model.

5.9.5 Temperature Calculations

The execution of the model for a given operating point follows the procedure outlined in Figure 5.14. The two remaining parameters to calculate are the discharge temperature of the compressor and the temperature of the lumped mass.

In order to calculate the discharge temperature, the revolution model is run a sufficient number of times such that the temperatures, pressures, and oil mass fractions at the beginning and end of the rotation are equal. All of these calculations are carried out for a given discharge temperature. After convergence has been achieved an updated discharge temperature is calculated. First the mean discharge enthalpy is calculated from

$$h_{disc} = \frac{\sum_{\text{discharge}} \dot{m}h}{\sum_{\text{discharge}} \dot{m}} \tag{5.101}$$

and the summations in Eqn. (5.101) are carried out over all the flow that enters and exits the discharge port. Over most of the rotation the flow will tend to exit the compressor, but there may be parts of the rotation where there is backflow into the compressor due to undercompression. For the points with backflow, the enthalpy h in Eqn. (5.101) is equal to that of the discharge downstream of the compressor. Finally a new updated discharge temperature is obtained by solving for the temperature as a function of the discharge enthalpy, pressure and oil mass fraction. An iterative method is used to drive the old and new values for the discharge temperature to the same value.

The lumped mass temperature is obtained by using a solver to drive the residual of the energy balance over the lumped mass r_{HT} from Eqn. (5.43) to zero. A range of numerical solvers have been tested, and using a secant solver for the calculation of the lumped mass temperature seems to work most of the time, though for some points a more robust Dekker solver is required.

Once both the discharge and lumped mass temperatures have converged, the model execution is stopped and all of the relevant outputs are written to file.

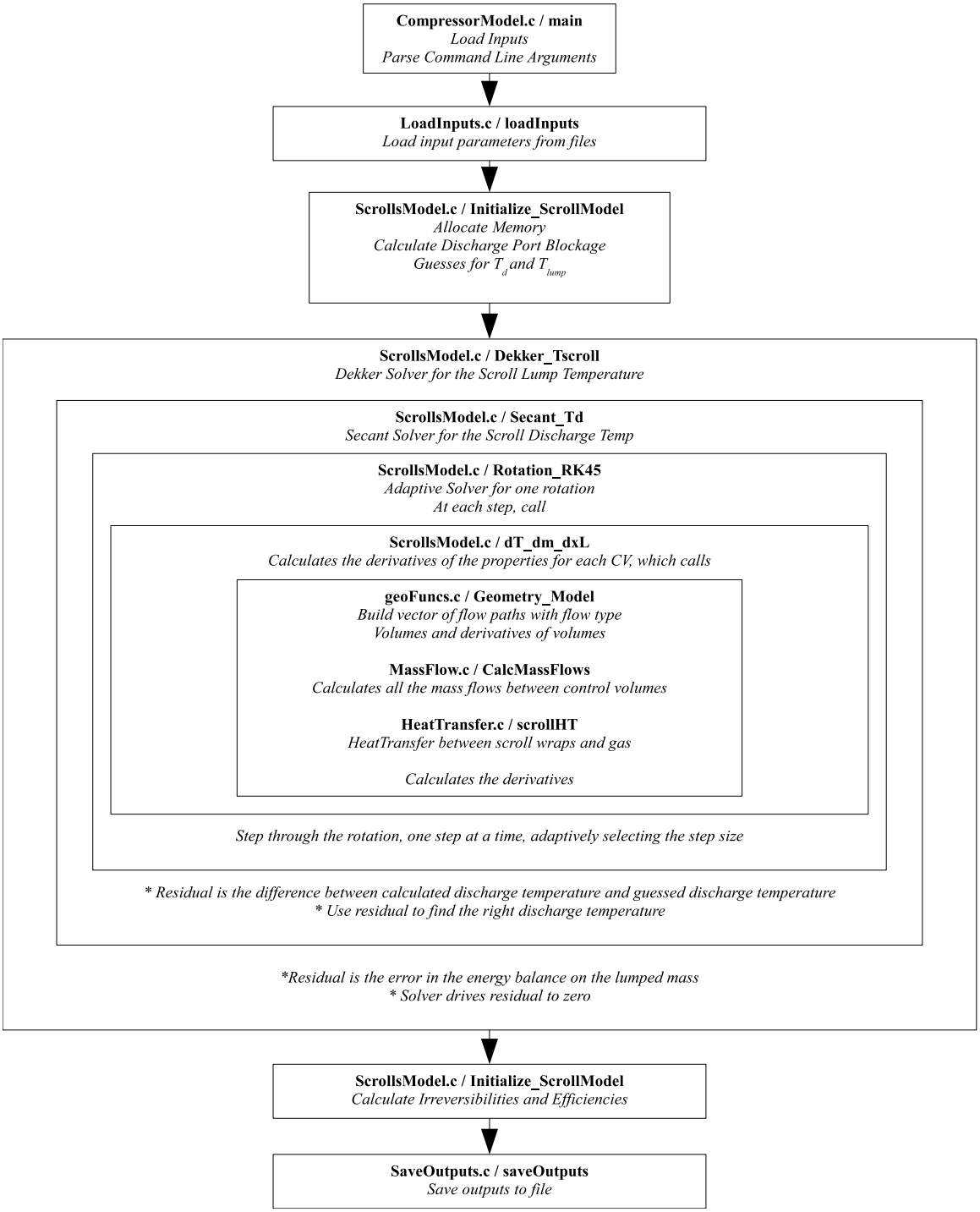


Figure 5.14. Flowchart of model execution.

5.10 Scroll Compressor Working Processes

The working processes of the flooded compressor are quite similar to those of a conventional compressor. For that reason, the working processes of the conventional compressor without liquid-flooding will be explained and then contrasted with the working processes of the scroll compressor with liquid-flooding. For both flooded and dry operation, the compressor is run at a constant rotational speed of 3500 revolutions per minute, and the temperatures and pressure are set to be near the mean values from experiments. The compressor modeled here is the Sanden compressor discussed further in Chapter 6.

5.10.1 Conventional Compression

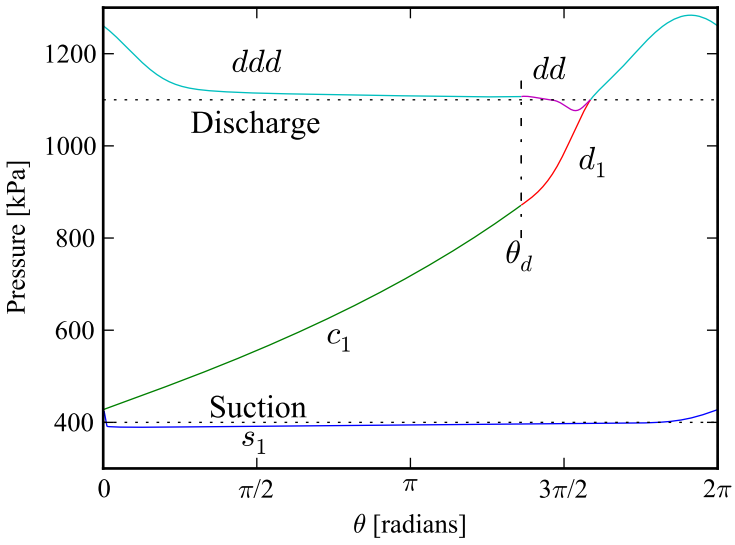


Figure 5.15. Pressures in the scroll compressor without flooding (nitrogen, p_s =400 kPa, p_d=1100 kPa, T_s=310 K, x_l=0).

Ultimately pressure is the thermodynamic property that can be thought of as driving the compressor. The pressure applied over the scroll wraps causes torques that the motor must overcome to compress the refrigerant. Pressure is also the motive force

that drives flow from one control volume to another, so to understand the ma ss flow, it is also important to understand the pressure evolution in the compressor. Figure 5.15 shows the pressures in the chambers of a conventional compressor without oil flooding. Beginning with the suction chamber, the suction chamber quickly takes on the compression chamber pressure because it begins with an infinitely small volume, and there is a relatively large leakage gap between the suction chamber and the compression chamber. The adaptive solver takes many very small steps during this part of the rotation due to the high stiffness of the system of equations. As the suction process proceeds, the volume of the suction pocket increases, which draws gas into the suction pocket from the suction line, and the small pressure drop between the suction line and the suction pocket is needed to drive gas into the suction pocket. Once the maximum volume of the suction chamber is reached, the suction chamber volume begins to decrease before the end of the rotation. This decrease in volume results in an increase in pressure - to above the pressure of the suction line in this case. The volumetric efficiency of the compressor can be above 100% if the density of the fluid in the suction pocket at the end of the rotation is above that of the suction line. Leakage and heat transfer to the gas in the suction pocket will tend to increase the temperature and decrease the density.

After the end of one rotation, the suction chamber becomes a compression chamber which is "pinched off" and can only communicate with other chambers through leakage. This compression chamber has a linear decrease in volume with the crank angle, and as a result has a monotonically increasing pressure until the discharge angle θ_d is reached.

At the discharge angle, the compression chambers are now treated as being part of the discharge region. The remaining volume shortly before the discharge angle is grouped back into the *dd* chamber. The d_1 and d_2 chambers equalize in pressure with the *dd* chamber, and once they have reached pressure equilibrium, the chambers are merged into the *ddd* chamber, which continues onwards to the discharge angle of the next rotation. It is possible that the discharge angle occurs just shortly before the

end of one rotation, in which case the merging process occurs at the beginning of the next rotation.

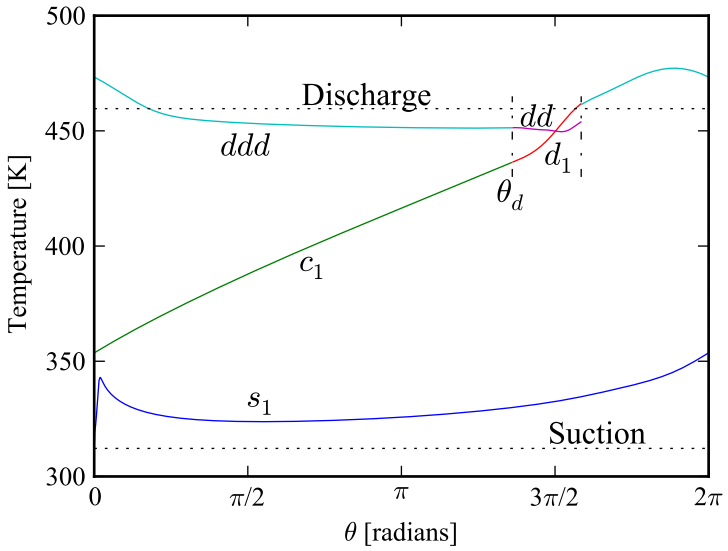


Figure 5.16. Temperatures in the compressor without flooding (nitrogen, p_s =400 kPa, p_d=1100 kPa, T_s=310 K, x_l=0).

The temperatures follow similar profiles to the pressures. Figure 5.16 shows the temperature profiles in each of the working chambers. The suction chamber begins with a temperature above the suction line due to pre-heating in the compressor inlet, but otherwise exhibits a temperature profile inline with its pressure profile. The compression pocket also exhibits a temperature profile like its pressure profile. During the discharge process, the pressures equalize, but the temperatures do not. There is no driving potential that would tend to drive the discharge temperatures together - short of heat transfer which has time constants far too large to be of use. This is the reason why the energy balance needs to be used to carry out the merging, described above.

A final point on the conventional scroll compressor regards the gas forces and torque. Over the course of a rotation, the gas applies a distributed load on the scroll faces, and the magnitude and direction of the radial force vector of each of the

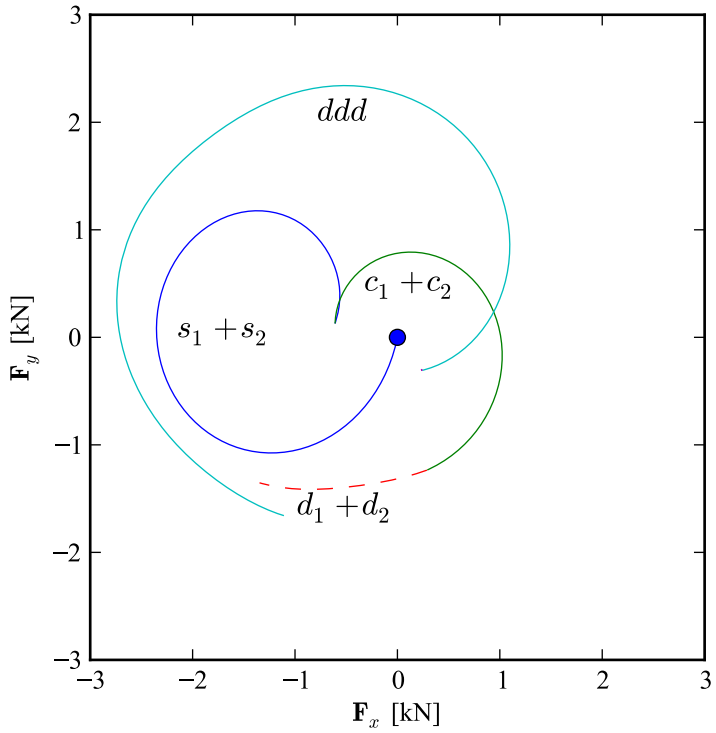


Figure 5.17. Radial force components in the scroll compressor without flooding (nitrogen, p_s =400 kPa, p_d=1100 kPa, T_s=310 K, x_l=0).

chambers changes with the crank angle. Figure 5.17 shows the trajectories of the radial force components generated by each of the control volumes. The distance of a point from the origin is equal to the magnitude of the force vector. The suction chamber begins with no volume, and as a result generates no radial force. As it fills with fluid it begins to generate a radial force and at the end of the rotation, it becomes a compression chamber. The force vector at the end of the rotation as a suction chamber and the beginning of the rotation as a compression chamber is continuous but does not have a continuous first derivative. This unsmooth behavior is counter-intuitive, but is related to the bounding involute angles which forms the control volume, and therefore the arc over which the pressure applies its force. In the course of the suction process, the length of the arc forming the inner surface of the chamber is constantly increasing up the point where it becomes a compression

chamber, at which point the length of the involute immediately begins to decrease. This abrupt change is reason for the discontinuous derivative of the force vector.

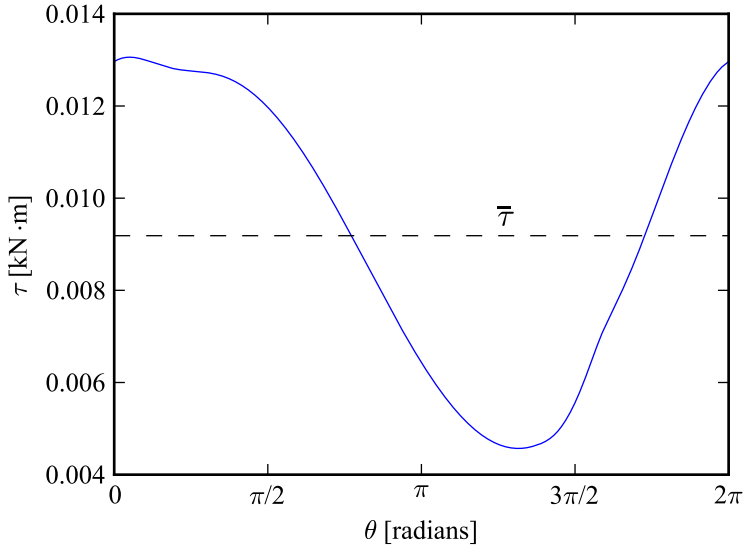


Figure 5.18. Torque generated by gas forces in the scroll compressor without flooding (nitrogen, p_s =400 kPa, p_d=1100 kPa, T_s=310 K, x_l=0).

From the stand point of predicting compressor performance, the torque is ultimately the most important parameter. Figure 5.18 shows the instantaneous torque generated by the gas forces from each control volume. The torque calculations are based on the analysis shown above. The scroll compressor exhibits a relatively even torque from the gas loading that is always positive and varies no more than 50% from the mean value shown by $\overline{\tau}$.

The adaptive solver uses a large range of step sizes over the course of the rotation that span nearly five orders of magnitude. Figure 5.19 shows the step size employed by the adaptive solver over the course of a rotation. The step size is very small at the beginning of the process due to the relatively large leakage area for the suction chamber relative to its volume, and then broadly increases in size over the course of the rotation. The second decrease in step size after a crank angle of $3\pi/2$ is due to the

merging process. There is a large flow area between the chambers, and the volume of the discharge chambers decrease, resulting in numerical stiffness.

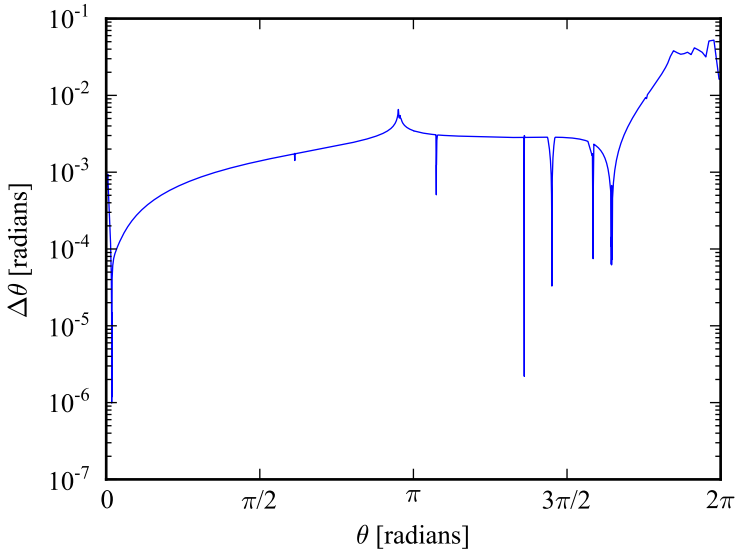


Figure 5.19. Step size for the scroll compressor without flooding (nitrogen, p_s =400 kPa, p_d=1100 kPa, T_s=310 K, x_l=0).

5.10.2 Flooded Compressor

Figure 5.20 shows the pressures in the scroll compressor when oil flooding with Zerol oil is employed, for a large oil flow rate of 80% oil by mass. When compared with Figure 5.15, it is clear that adding the oil has the tendency to increase the pressure drop in the suction and discharge processes and increase the under-expansion losses. The suction and pressure drops are due to the increased mass flux passing through the ports, causing large pressure drops.

With regards to the under-expansion, for the same volume ratio, the mixture sees a smaller increase in temperature and pressure because of the mixing of gas and oil. As a result, the optimal built-in volume ratio for oil-flooded compressors is greater than for conventional compressors. The analysis of Chapter 7 develops a simple model for the optimal built-in volume ratio for oil-flooded compressors. For the conventional

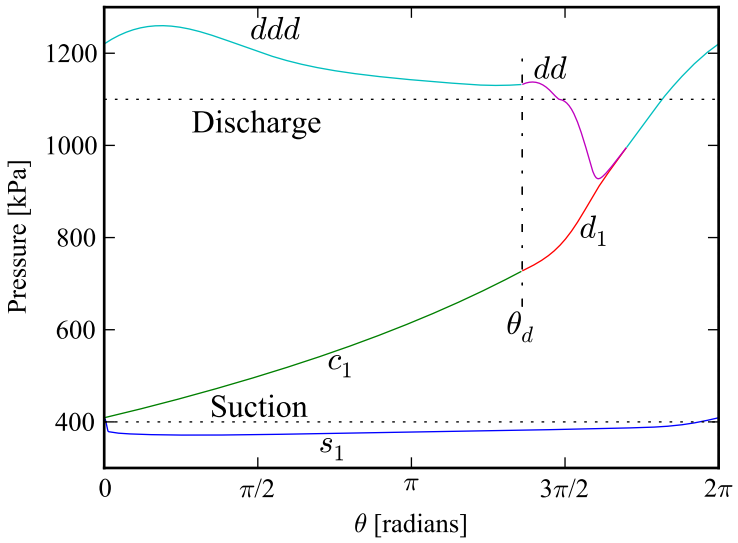


Figure 5.20. Pressures in the scroll compressor with flooding (nitrogen, Zerol oil, p_s =400 kPa, p_d=1100 kPa, T_s=310 K, x_l=0.8).

compressor, the pressure in the compression chamber right at the discharge angle is 868.2 kPa while that in the flooded compressor is 726.7 kPa.

The temperatures in the flooded compressor increase much less than for the conventional compressor over the course of the rotation. The discharge temperature decreases from 458.6 K to 325.6 K when oil flooded. Figure 5.21 demonstrates the shape of the temperature profile, which is similar to that of the conventional compressor except that the magnitudes of the temperatures are all scaled downwards.

The oil mass fraction varies quite significantly over the course of the rotation. Since the leakage model is based on the flow of only gas through the leakage gaps, net leakage into a control volume will tend to decrease the oil mass fraction, and net leakage out of a control volume will tend to decrease the oil mass fraction. Figure 5.22 shows the oil mass fractions in the control volumes over the course of a rotation. At the beginning of the suction process the volume of the s_1 chamber is very small relative to the leakage area between it and the c_1 chamber. As a result, a large amount of gas can leak back to the suction chamber, which results in a significant

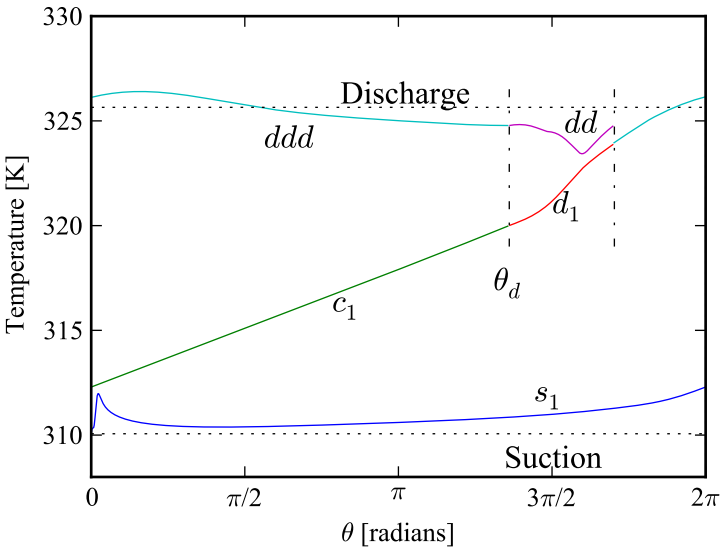


Figure 5.21. Temperatures in the scroll compressor with flooding (nitrogen, Zerol oil, p_s =400 kPa, p_d=1100 kPa, T_s=310 K, x_l=0.8).

dilution, and a rapid decrease in oil mass fraction initially. As the primary flow with a significant amount of oil begins to enter the suction chamber, the oil mass fraction increases, and then as leakage again begins to play a larger role due to the larger flow area, the oil mass fraction begins to decrease again. The compression chamber's oil mass fraction tends to decrease since it experiences a net leakage in. Finally in the discharge region, the discharge chambers leak to the lower pressure chambers, which results in a net increase in oil mass fraction.

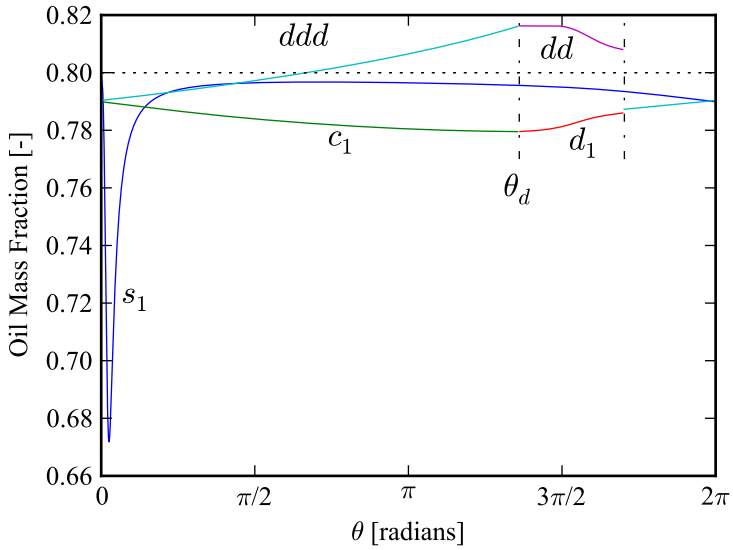


Figure 5.22. Oil mass fractions in the scroll compressor with flooding (nitrogen, Zerol oil, p_s =400 kPa, p_d=1100 kPa, T_s=310 K, x_l=0.8).

CHAPTER 6. EXPERIMENTAL STUDY OF LIQUID-FLOODED ERICSSON CYCLE AND MODEL VALIDATION

6.1 Experimental Setup

A scroll compressor and scroll expander were installed in a test rig in order to measure flooded Ericsson cycle performance as seen in Figure 6.1. A more detailed schematic is available in Appendix D. The primary goal of this experimentation with the Ericsson cycle testing was to provide a data set which could be used to validate the compressor model as well as provide further information about the use of flooded compression and expansion.

Beginning a description of the system with the compressor, oil and gas are adiabatically mixed at state point 21. The oil and gas are compressed together in the compressor from state point 22 to state point 23, at which point the oil-gas mixture passes into the hot heat exchanger at state point 29 and is cooled to state point 30. The mixture is cooled against an ethylene glycol-water temperature bath. After exiting the hot heat exchanger, the two-phase mixture enters into the hot-side separator (state point 26) where the oil and gas are separated into oil (state point 31) and gas phases (state point 32). The oil is then expanded from high pressure (state point 24) to low pressure (state point 25) in a hydraulic expander to generate electrical power, and mixed back into the hot gas stream (state point 19). The hot gas exiting the separator then enters the regenerator (state point 8) where it is cooled to state point 9.

After exiting the regenerator, the cooled gas (state point 9) is mixed with cool oil (state point 3) to state point 5. This two-phase mixture enters the expander (state point 6) where it is expanded to state point 7. After the expansion process, the two-phase mixture passes into the cool heat exchanger, where the mixture is heated

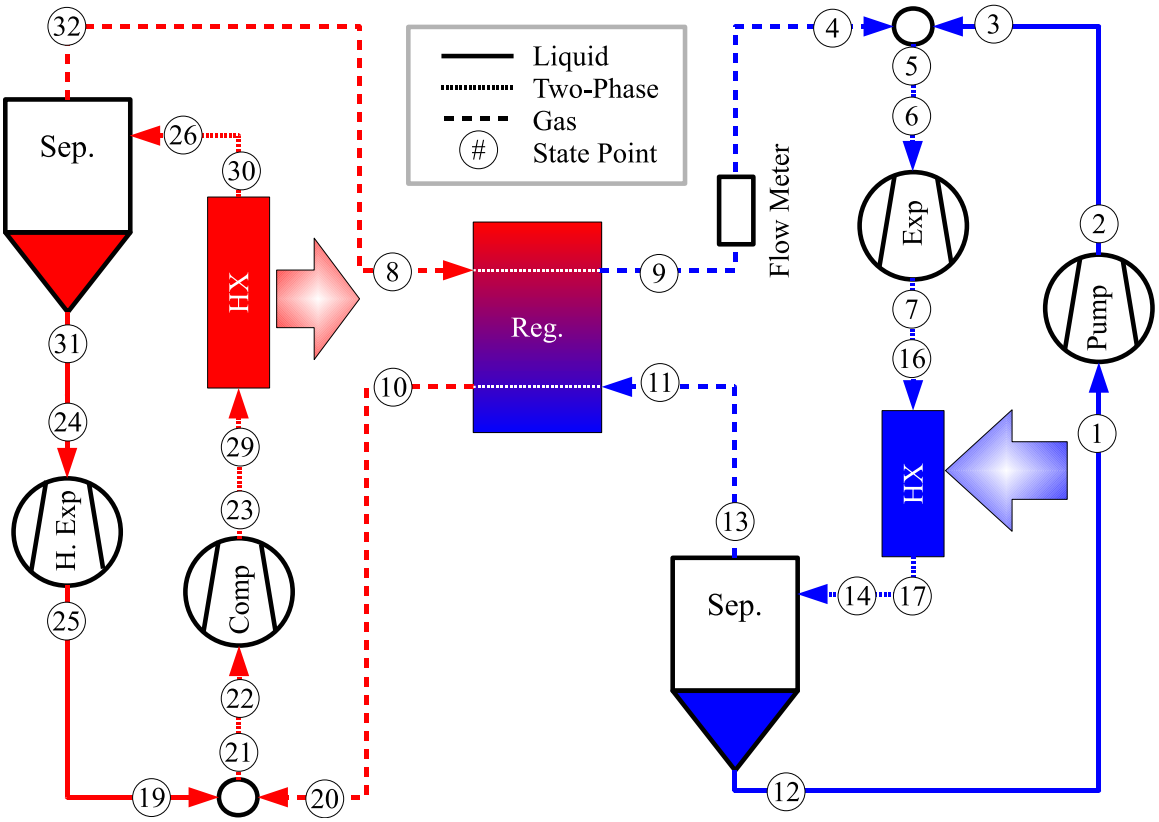


Figure 6.1. System Configuration.

to state point 17, providing the cooling capacity of the Ericsson cycle. The heated two-phase mixture enters into the cool separator at state point 14, and is separated into oil (state point 12) and gas streams (state point 13). The oil stream is pumped up from low pressure (state point 1) to high pressure (state point 2), and then mixed back into the gas stream. The gas exiting the cold separator enters the regenerator where it is warmed from state point 11 to state point 10.

6.1.1 Components And Measurements

Compressor/Expander

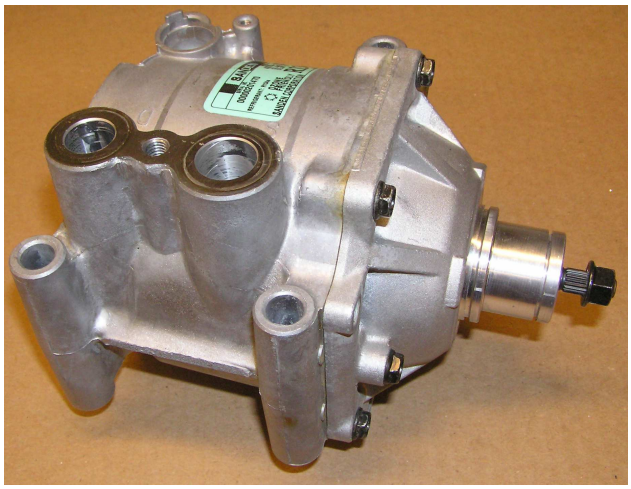


Figure 6.2. The Sanden model TRS-105 scroll compressor used in this study.

Both the compressor and expander were Sanden automotive compressors model TRS-105 shown in Figure 6.2, with compressor suction displacement of 104.8 cm^3 per revolution. The displacement of the expander is the compressor displacement divided by the built-in volume ratio, which has the value of 1.61 as measured from the scroll compressor profile. Hugenroth (2008) provides a contradictory value for the volume ratio of 1.8. He claims that this volume ratio was obtained from coordinate measurement machine data, but the details of how this value was obtained are not available.

The discharge valves from both machines were removed, and the radial compliant mechanisms for both compressors were pinned to disable the radial compliance.

The compressor employs a system of ball bearings to enforce the orbiting motion as well as support the axial load. The ball bearings run in a system of two raceways which bear simultaneously on the ball bearings in two directions, seen in Figure 6.3 with the ball bearings removed. The distance between the contact points on each ball is equal to the twice the orbiting radius of the compressor. This figure also shows the counter-weights and the radial ball bearing.

Figure 6.3. Internal structure of scroll compressor bearing system.

Oil Separators

The oil separators provide for a volume in which the oil and vapor can separate into two independent phases. Overall the separators seem to have been undersized for the large flow rates of oil passing through the system. As a result of being undersized, the oil flow contained some amount of entrained gas bubbles, which was noted as a problem by Hugenroth (2006), and remained a problem through the testing conducted here.

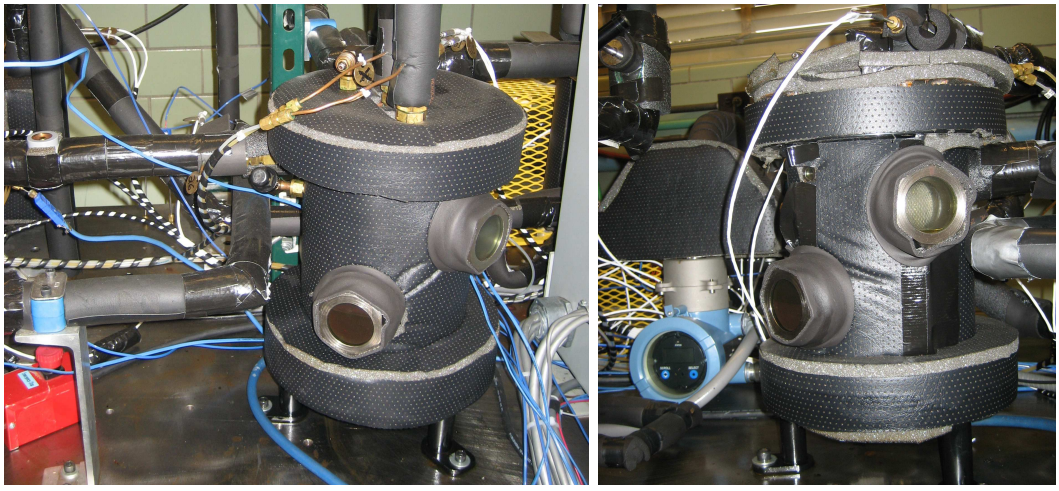
A two-phase mixture of oil and gas enters the oil separator tangentially. If the separator is operating ideally, completely separate oil and gas streams exit the separator. A dip tube is used in the separator to return only oil in the oil lines.

The same separators were used on both the hot and cold loops of the system and were custom manufactured to suit this application. The internal dimensions of the separators were an internal diameter of 11.27 cm (5-9/16") and an internal height of 24.1 cm (9-1/2"). They are shown in Figure 6.4. The internal structures of the separator were constructed to assist in the gravitational separation. Figure 6.5 shows the mesh structure that was placed on the inside of the separator. There are two layers of mesh - a coarse mesh for structure and a fine mesh for better coalescing performance. Oil and gas enter through the hole in the spiral shown in the bottom right of the photo. The inwardly spiraling path forces the oil to contact the mesh, which causes the small oil droplets to coalesce and fall towards the bottom of the separator. The mesh structure is inserted upside down into the top of the oil separator.

Secondary separators (Henry Technologies model S-5185) were installed downstream of the primary oil separators in the gas line to separate out any remaining oil in the gas line, though they did not seem to be removing much, if any, oil from the gas stream.

Hydraulic Expander/Pump

The hydraulic pump is a Sidener Engineering model T6CM-B05-1L01-C5 with displacement of 17.21 cm^3/rev (1.05 in^3/rev). The hydraulic expander is a Sidener Engineering model M5BS-018-1N02-A502 with displacement of 18.03 cm^3/rev (1.10 in^3/rev). The speeds of the hydraulic expander and pump were controlled using a variable speed motor controller. One major limitation of the hydraulic pump was that the suction pressure could not exceed 689 kPa (100 psig). This meant that the standstill pressure of the system could not exceed 689 kPa, so to get to higher operating



(a) Hot oil separator (b) Cold oil separator

Figure 6.4. Primary oil separators for hot and cold loops.

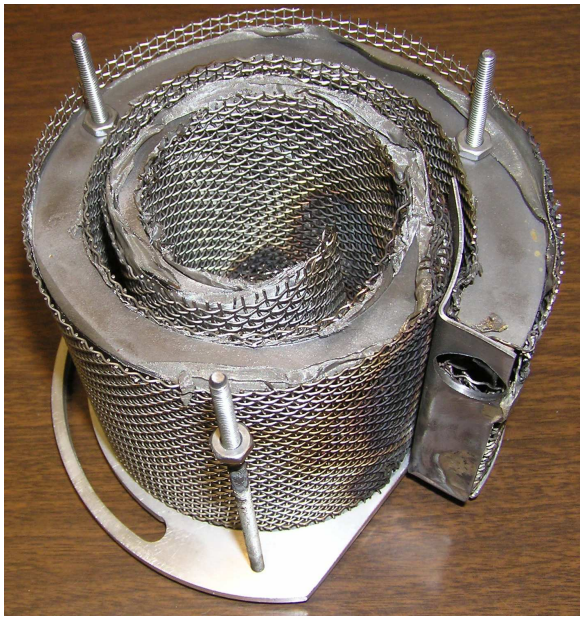


Figure 6.5. Internal structure of oil separator (Shown upside down).

system pressures with effective stand-still pressures above 689 kPa, nitrogen had to be added while the system was operating, and removed prior to shut-down.

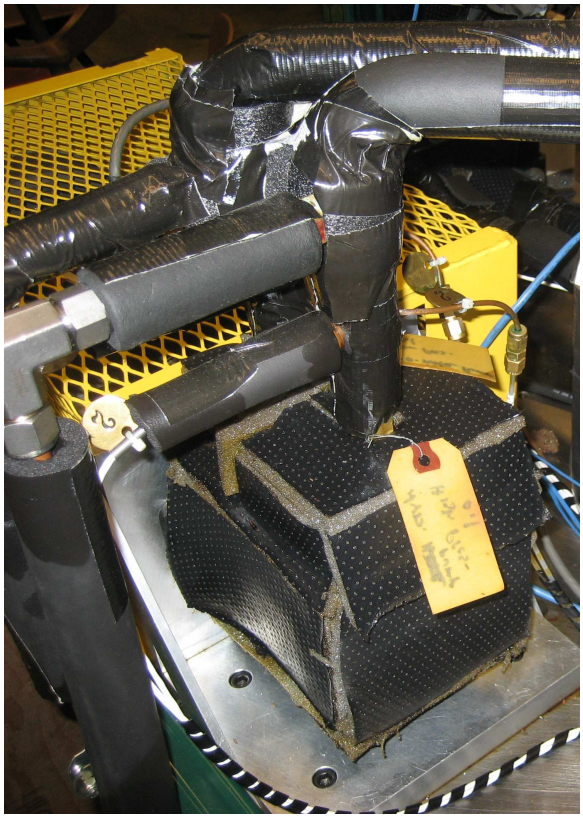


Figure 6.6. Hydraulic Pump.

Heat Exchangers

The hot and cold heat exchangers were FlatPlate brazed-plate heat exchangers; both hot and cold side heat exchangers were model FP5x12L-14, and the regenerator was a model FP5x12-20. The regenerator and hot heat exchanger were properly piped in counterflow, but the cold heat exchanger was piped in parallel flow, which likely decreased the effectiveness of the cold-side heat exchanger.

Glycol Loop

A glycol chiller was employed to maintain a glycol bath at a constant temperature. The glycol from this bath served as both the heat source for the cold loop as well as the heat sink for the hot loop. Control of the glycol supply temperature was achieved

by using a fixed-capacity chiller and a variable capacity electric heater to balance the chiller. Pure Dowtherm SR1 was circulated as the working fluid in the glycol loop to decrease uncertainty about the percentage of glycol in solution. A positive displacement gear pump was used to pump glycol from the chiller through the glycol lines.

Measurement Devices

The gas flow was determined using a Coriolis mass flow meter, and both oil separator gas outlet flows were seen to be very dry and assumed to be liquid-free. Due to thermal non-equilibrium for the heat exchanger flows, the liquid mass flows was calculated from energy balances over the compressor and expander as described below. The mass flow of the aqueous ethylene glycol mixture used as a heat transfer fluid in the heat exchangers was also calculated using a Coriolis mass flow meter.

The shaft power of each rotating component was measured with the use of Sensor Developments model 01324-022 rotary torque cells with a full range of 22.6 N-m (200 in-lbf). The rotational speed of the rotating components was obtained from the motor controllers which output the rotational speed with an accuracy of 1 rev/min.

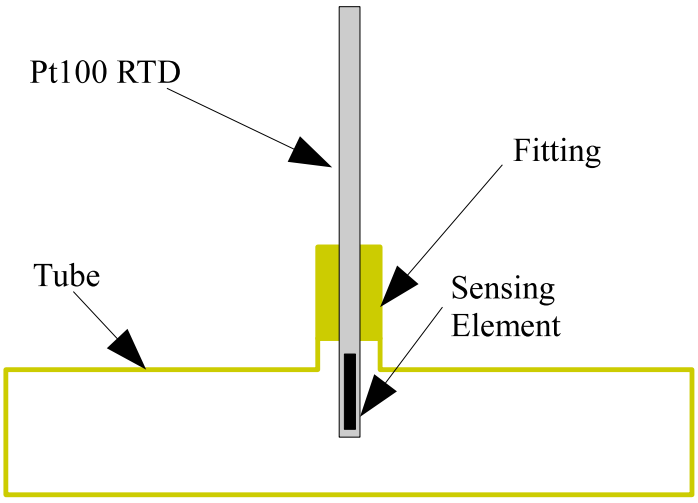


Figure 6.7. Schematic of RTDs installed perpendicular to the tube.

The temperature at all points of the cycle were initially measured with 4-wire Pt100 resistance thermometer devices (RTD) because one of the recommendations of Hugenroth (2006) was that the uncertainty in temperature measurement should be decreased as the temperature uncertainty contributed significantly to the overall uncertainty. Unfortunately, when the test rig was rebuilt by the manufacturer, all the RTDs had been installed perpendicular to the tubes with a large length of RTD exposed into the ambient like the configuration shown in Figure 6.7. The problem with this configuration is that the stem conduction is no longer negligible because the length between the element and the fitting is relatively small. This is why RTDs should be installed fully immersed in a T-junction in order to increase the stem conduction thermal resistance. The area of the tip that the RTD averages to yield the measured temperature is greater than that of thermocouples which yield something much closer to a point-measurement since the actual temperature sensor is just the junction of two wires. Unfortunately, due to the large number of temperature measurements, it was not possible to correct the installation problems.

In order to partially alleviate the temperature measurement challenges at the compressor and expander, some of the RTDs were replaced with T-type thermocouples. Even so, large challenges in temperature measurement were found due to the thermal non-equilibrium effects described below. The pressures in the system were measured with Setra model 207 pressure transducers, with full scale ranges of 0-17.23 bar gage (0-250 psig) for the low pressure measurements, and 0-34.47 bar gage (0-500 psig) for the high pressure measurements. A mercury barometer was used to measure the ambient pressure. The measurement devices are summarized in Table 8.1.

6.1.2 Test Matrix

Tests were carried out over the 27 points formed as the permutations of the points in the test matrix shown in Table 6.2. The experimental data from all the tests carried out (including additional tests) is in Appendix D.

Table 6.1 Summary of measurement devices and uncertainties.

Measurement	Device	Uncertainty
Temperature	Pt100 RTD & T-Type TC	0.8 K [a]
Low-Side Pressure	Setra Model 207	2.24 kPa
High-Side Pressure	Setra Model 207	4.48 kPa
Mass Flow Gas	MicroMotion Model R025	1.0%
Mass Flow Glycol	MicroMotion Model F050	0.2%
Rotational Speed	Baldor	1 RPM
Shaft Torque	Sensor Developments 01324-022	0.0452 N·m

[a]Includes both sensor uncertainty and the errors generated from thermal non-equilibrium effects. Even so, likely underestimates temperature measurement error.

Table 6.2 Test Matrix for testing of Liquid-Flooded Ericsson Cycle.

Parameter	Nominal Values Used
Compressor/Expander Speed Ratio	2, 3, 4
Compressor Inlet Pressure (gage)	238, 375, 414 kPa
Hydraulic Expander/Pump Speed	120/60, 240/120, 480/240 RPM

In order to achieve a given target operating point, the speeds of the compressor, expander, hydraulic pump and hydraulic expander were slowly ramped up to their desired values. The compressor was always run at a rotational speed of 3500 RPM. Nitrogen was added or removed from the system in order to achieve the desired compressor suction pressure. For the highest compressor inlet pressure, the resting pressure is above the maximum pressure for the hydraulic pump suction. Thus, to achieve a compressor inlet pressure of 414 kPa gage, charge had to be added while the system was running and removed before the system could be shut down. This made the operation for the 414 kPa gage compressor inlet pressure point more complicated than necessary. If the Liquid Flooded Ericsson Cycle is to be further

investigated experimentally, it would be beneficial to find a hydraulic pump with a higher maximum suction pressure.

Some control over the compressor and expander inlet temperatures was available by altering the glycol bath temperature and the flow rates of glycol through the hot and cold heat exchangers, but in general, the compressor and expander inlet temperatures were dependent on the other inputs.

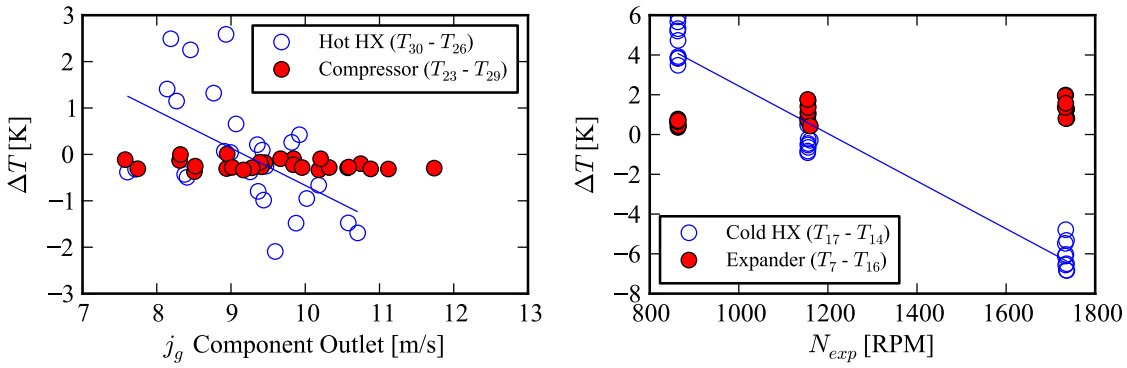
The system was run until steady-state operation was achieved, which took on the order of an hour to reach a quasi-steady condition. It was impossible to fully reach a steady-state condition due to the extremely large amount of thermal mass in the system. All the rotating components were solidly bolted into the table, so for the system to achieve steady temperatures, the table must also reach steady-state. Even after more than 10 hours straight of running the load stand, the temperature of the table never stabilized.

6.1.3 Measurement Of Oil Flow Rate

Initially it was desired to use the energy balance over the hot and cold heat exchangers to determine the hot and cold oil flow rates, but it was found that at certain points in the system, there was a significant amount of thermal non-equilibrium between the gas and liquid phases. For instance, the temperature difference between the outlet of the hot heat exchanger and the inlet to the hot separator, over a distance of about a meter, ranges between -2.1 K and 2.6 K, as seen in Figure 6.12(c). The temperature differences on the hot side are plotted against the superficial gas velocity which is typically used to characterize flow patterns in two-phase flow. The superficial gas velocity is defined by

$$j_g = \frac{\dot{m}_g}{\rho_g A} \tag{6.1}$$

where A is the cross-sectional flow area of the tube, where the tubes with two-phase flow are all 3/4" copper tubes with inner diameter of 16.9 mm. The density of the gas ρ_g is evaluated at the temperature and pressure right at the outlet of the component.



(a) Hot side

(b) Cold side

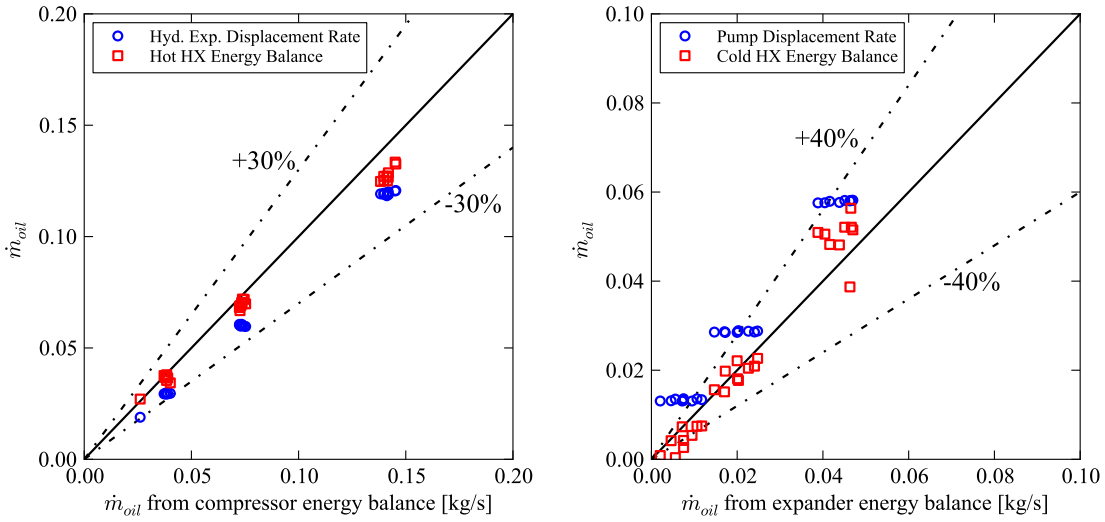
Figure 6.8. Temperature differences downstream of components.

Figure 6.8(b) shows that the same problem is also manifested at the cold-side heat exchanger where the temperature differences range from -6.8 K to +5.9 K. Here the temperature is better correlated with the rotational speed of the expander. Because of the large temperature differences downstream of the heat exchangers it is challenging to approximate the oil flow rate based on a heat exchanger energy balance.

The differences in temperature are due to the differences in flow pattern and inter-facial heat transfer inside the components. For the compressor, the maximum absolute difference in temperature is 0.4 K, and for the expander the maximum absolute temperature difference is 2.0 K. Therefore a decision was made to use the energy balance over the compressor and expander to back-calculate the oil flow rates through the hot and cold sides of the rig.

The mass flow rate through the compressor and expander can be calculated by three different methods and compared. The first method is based on the displacement rate of the hydraulic pump (cold loop) or hydraulic expander (cold loop). This method assumes that the volumetric efficiency of the device is 100% and that there are no gas bubbles in the oil which was shown visually to be a bad assumption. A small amount of bubbles by mass has a very large impact on the mass flow rate of oil. A second method of calculating the oil flow rate is to use an energy balance over the

cold and hot heat exchangers to back out the oil flow rate, but the problems with this method were described above. Finally the oil flow rate can be obtained by an energy balance over the compressor and expander. Figure 6.9 shows the results of the three calculation methods over the 27 state points employed. There is a large amount of variation between the calculation methods, but for the compressor, all three methods result in relatively similar results. The results for the expander have a much larger range of results, partly due to the lower oil flow rates and what appears to be a larger amount of thermal non-equilibrium effects.



(a) Hot side

(b) Cold side

Figure 6.9. Calculation of oil flow rate by various methods.

In addition, from an experimental uncertainty standpoint, using the compressor energy balance is preferable because the heat exchanger energy balance requires four temperature measurements while the compressor energy balance only requires two. The dominant source of uncertainty in the heat balance is temperature measurement, so the fewer temperature measurements needed the better.

6.1.4 Data Reduction

For each machine, the shaft power is given by

$$\dot{W}_{shaft,meas} = N\frac{2\pi}{60}\tau \tag{6.2}$$

where the rotational speed N is given from the VFD, and the value of the measured torque τ is taken to be positive for the compressor and hydraulic pump and negative for the expander and hydraulic expander. Thus the energy balance for both the compressor and expander are given by

$$\dot{W}_{shaft,meas} = \begin{bmatrix} \overline{UA}_{amb}\left(T_{shell} - T_{amb}\right) - \dot{m}_l\left(h_{l,out} - h_{l,in}\right) \\ -\dot{m}_g\left(h_{g,out} - h_{g,in}\right) \end{bmatrix} \tag{6.3}$$

where the value of the shell temperature T_{shell} is based on the inlet temperature for the compressor and the outlet temperature for the expander. The inlet and outlet enthalpies of liquid and gas are known from measurements of inlet and outlet temperature and pressure. Thus the only remaining parameter needed to calculate the mass flow rate of oil is the overall shell-ambient heat transfer. Both compressor and expander are entirely covered with approximately 1 cm of foam insulation, and from a simplified network heat transfer analysis, the value of $\overline{UA}_{amb}$ is approximated as 0.001 kW/K. Thus the oil flow rate can finally be obtained from Eqn. (6.3). The oil mass fraction at the inlet of the compressor and expander can be given by

$$x_l = \frac{\dot{m}_l}{\dot{m}_l + \dot{m}_g} \tag{6.4}$$

where $\dot{m}_l$ is the mass flow rate predicted based on the energy balance on the compressor or expander.

The volumetric effectiveness of the scroll machines is defined based on the swept volume of the scroll machine. In the expander, the displacement volume is equal to the compressor suction volume divided by the built-in volume ratio. Thus the displacement volumes of the compressor and expander are 104.8 cm^3 and 65.5 cm^3 respectively. The volumetric effectiveness can be defined as

$$\eta_v = \frac{\dot{m}_{m,meas}}{\dot{m}_{m,ideal}} = \frac{\dot{m}_{m,meas}}{\rho_{m,in} V_{disp} \dfrac{N}{60 \text{ s/min}}} \tag{6.5}$$

where the volumetric efficiency of the compressor will be in general less than one, and the volumetric efficiency of the expander will in general be greater than one. The energy efficiency of the scroll machines is defined based on the overall isentropic efficiency of the machine, given for the compressor by

$$\eta_{oi,comp} = \frac{\dot{m}_{m,meas}(h_{out}|_{s=s_{in}} - h_{in})}{\dot{W}_{shaft,meas}} \tag{6.6}$$

where the enthalpies and entropies are based on mixture properties as described above. Similarly, the overall isentropic efficiency of the expander can be defined by

$$\eta_{oi,exp} = \frac{-\dot{W}_{shaft,meas}}{\dot{m}_{m,meas}(h_{out}|_{s=s_{in}} - h_{in})} \tag{6.7}$$

where $\eta_{oi,comp}$ and $\eta_{oi,exp}$ are both less than one.

The volumetric efficiencies of the hydraulic expander and hydraulic pump can be defined by

$$\eta_{v,pump} = \frac{\dot{m}_{l,cold}}{\rho_{in} V_{disp,pump} N_{pump} \dfrac{1}{60 \text{ s/min}}} \tag{6.8}$$

$$\eta_{v,hyd.exp.} = \frac{\dot{m}_{l,hot}}{\rho_{in} V_{disp,hyd.exp.} N_{hyd.exp.} \dfrac{1}{60 \text{ s/min}}} \tag{6.9}$$

where $\dot{m}_{l,hot}$ and $\dot{m}_{l,cold}$ are the oil flow rates based on energy balances carried out on the compressor and expander respectively. The hydraulic expander and hydraulic pump are assumed to be pumping an incompressible fluid, thus their overall isentropic efficiencies are given by

$$\eta_{oi,hyd.exp.} = \frac{\dot{W}_{shaft,hyd.exp.}}{\dot{m}_{l,hot} v_{l,in}(p_{25} - p_{24})} \tag{6.10}$$

$$\eta_{oi,pump} = \frac{\dot{m}_{l,cold} v_{l,in}(p_2 - p_1)}{\dot{W}_{shaft,pump}} \tag{6.11}$$

which are both less than one. The effectiveness of each of the heat exchangers is obtained from

$$\epsilon_{HX} = \frac{C_h(T_{h,i} - T_{h,o})}{C_{min}(T_{h,i} - T_{c,i})} \tag{6.12}$$

where C_h is equal to the product of the hot stream mass flow rate time the mean specific heat of the hot stream, and C_{min} is equal to the minimum capacitance rate of the hot and cold streams. The cooling capacity is measured on the glycol loop, and defined by

$$\dot{Q}_{cold} = \dot{m}_{gly} c_{p,gly} (T_{15} - T_{18}) \tag{6.13}$$

and the cycle COP is then given by

$$COP = \frac{\dot{Q}_{cold}}{\dot{W}_{shaft,comp} + \dot{W}_{shaft,exp} + \dot{W}_{shaft,hyd.exp.} + \dot{W}_{shaft,pump}} \tag{6.14}$$

which is the measure of the efficiency of the cycle.

6.2 Experimental Results

The fundamental goal of liquid flooding is to approach isothermal compression and expansion processes. As shown in Figure 6.10, the ratio of the high to low temperature for the compressor and the expander both approach 1.0 as the oil mass fraction increases, that is the process becomes more and more isothermal. In the

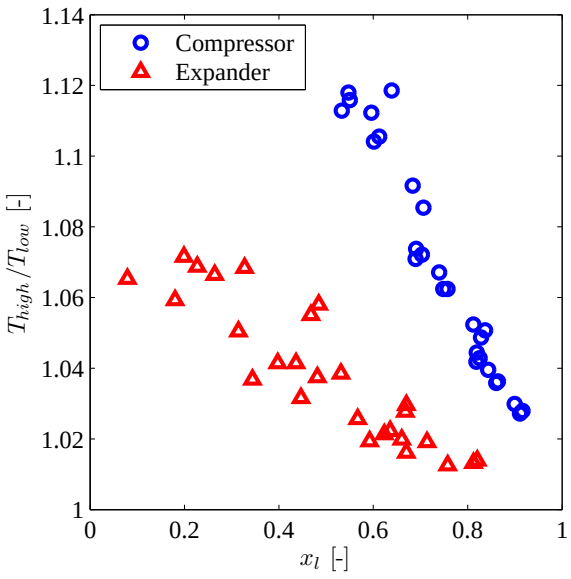
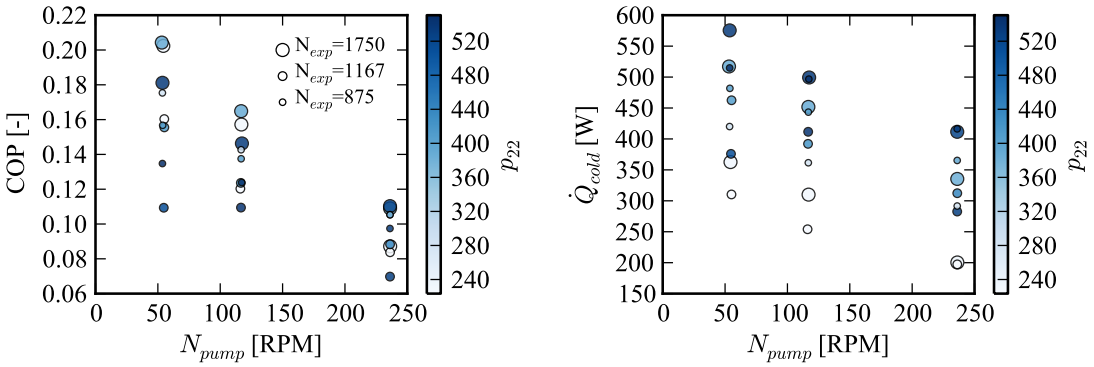


Figure 6.10. Experimentally measured temperature ratios for compressor and expander.

compressor, the high temperature is the outlet temperature, and in the expander, the high temperature is the inlet temperature. The difference in slope for the scatter plots for both compressor and expander is due to the difference in pressure ratios experienced by the two machines. Since the test rig is quite large with significant piping and a large number of fittings, the pressure drop between the compressor and expander is quite large. As a result, the imposed pressure ratio on the compressor will always be higher than that imposed on the expander. In the limit of no pressure drops in the system, the two curves should come significantly nearer. There will still be some difference in slope due to differences in scroll machine efficiency, manifesting itself as a difference in the outlet temperature.

From a cycle performance standpoint, the critical parameters are the cycle cooling capacity and the cycle coefficient of performance. From both of these standpoints, the performance of the Liquid-Flooded Ericsson Cycle is quite poor. Figure 6.11 shows the COP and capacity of the LFEC as a function of pump rotational speed, compressor suction pressure, and speed ratio. The best value of COP achieved is just slightly over 0.2, and the maximum capacity achieved is slightly over 550 W. In general the capacity is better for lower low-side pressures, and the system COP is better for lower-high side pressures. The addition of oil to both loops (shown by the pump rotational speed), tends to decrease both system COP and cooling capacity. In general, for conditions otherwise constant, both the capacity and COP increase with the rotational speed of the expander.

By comparison, a small window air-conditioner might have a cooling capacity of 1465 W (5000 Btu/h) and an EER of 10, corresponding to a COP of 2.93. The redeeming quality of the LFEC is that it uses environmentally-benign working fluids, but with such a low system efficiency, it will require significant amounts of primary energy to run the LFEC system for a given cooling capacity. Unless very low source temperatures are employed, the Liquid-Flooded Ericsson Cycle does not make much sense from a thermodynamic standpoint.

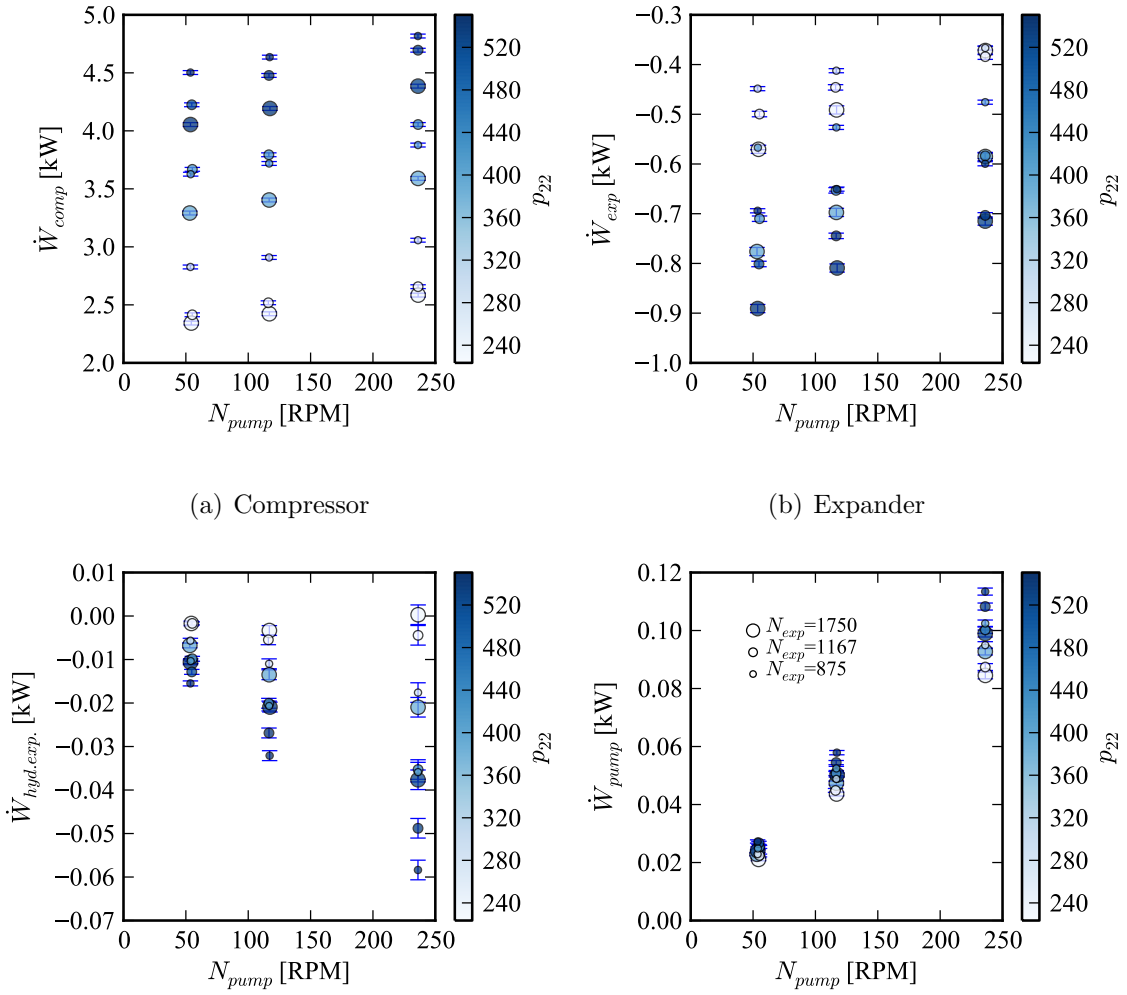


(a) System COP

(b) Cooling Capacity

Figure 6.11. COP and Capacity of LFEC as a function of pump rotational speed, expander rotational speed (marker size, where N_{exp} is rotational speed of expander in RPM), and compressor inlet pressure (marker color).

The primary thrust of this chapter regards the performance of the rotating machinery, in particular the scroll compressor. Results are presented for the other components as well in order to provide some insight into the performance of the components with flooding for future research. To begin with, the shaft powers for all the rotating machinery are presented because the shaft power has a quite low uncertainty and a number of clear parametric effects can be seen. Figure 6.12 shows the shaft powers of the components as a function of pump rotational speed (always half that of the hydraulic expander), compressor suction pressure, and expander speed. These independent variables are selected as they are the system parameters that were used to define the test matrix. In general, as the pressures in the system are increased, the compressor and expander have larger power consumption and generation respectively. For a given set of suction pressure and expander speed, the shaft power tends to increase with oil flow rate (or pump speed). The relative uncertainties of all the shaft powers are quite small, even for the hydraulic expander which has the smallest magnitude of shaft power.



(a) Compressor

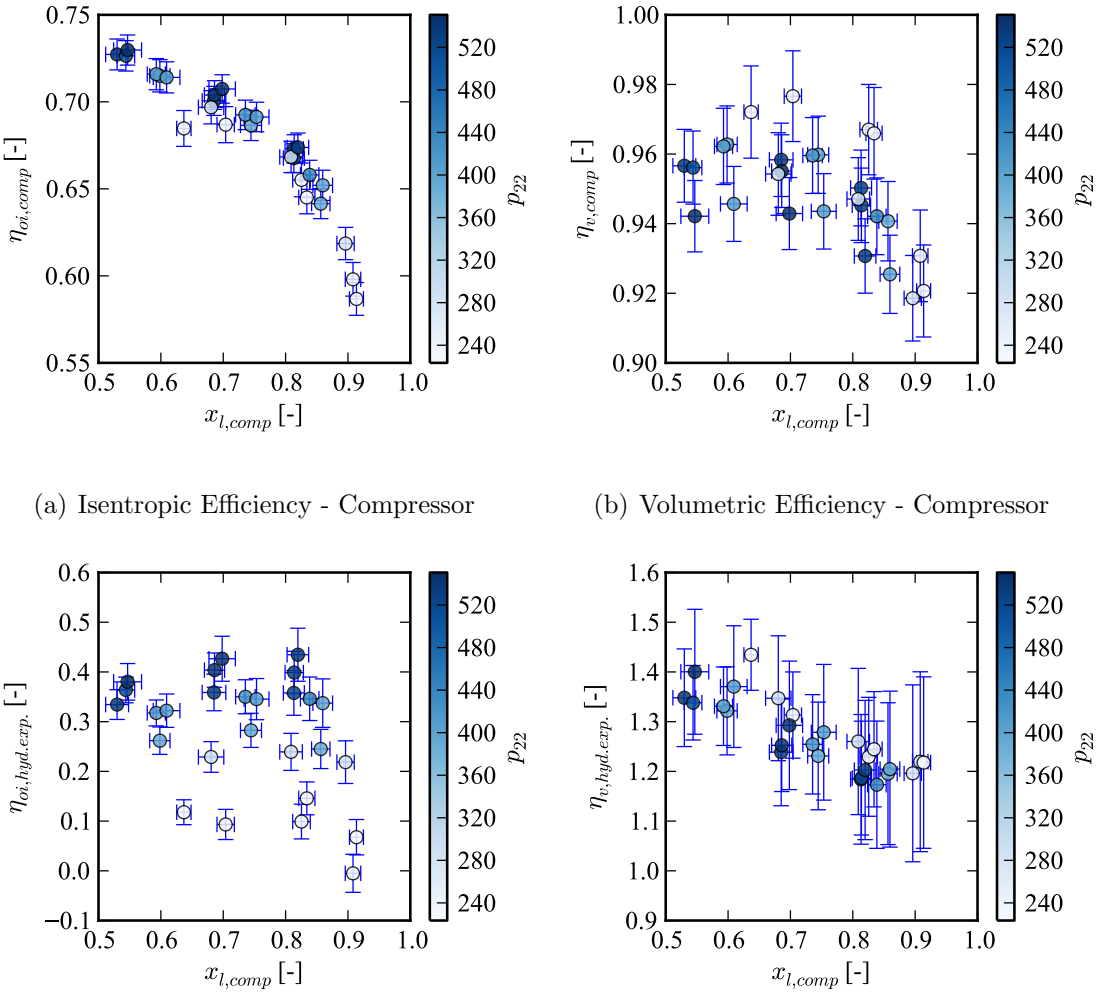
(b) Expander

(c) Hydraulic Expander

(d) Hydraulic Pump

Figure 6.12. Shaft powers of rotating components as a function of pump rotational speed, expander speed (marker size, where N_{exp} is rotational speed of expander in RPM) and compressor inlet pressure (marker color).

When the efficiencies of the components are considered, the trends are more difficult to visualize due to the scatter caused by the estimation of the oil mass flow rates from the energy balances on the compressor and expander. Figure 6.13 shows the overall isentropic and volumetric efficiencies for the compressor and hydraulic expander. For the compressor the trends are quite clear - as the oil mass flow fraction



(a) Isentropic Efficiency - Compressor

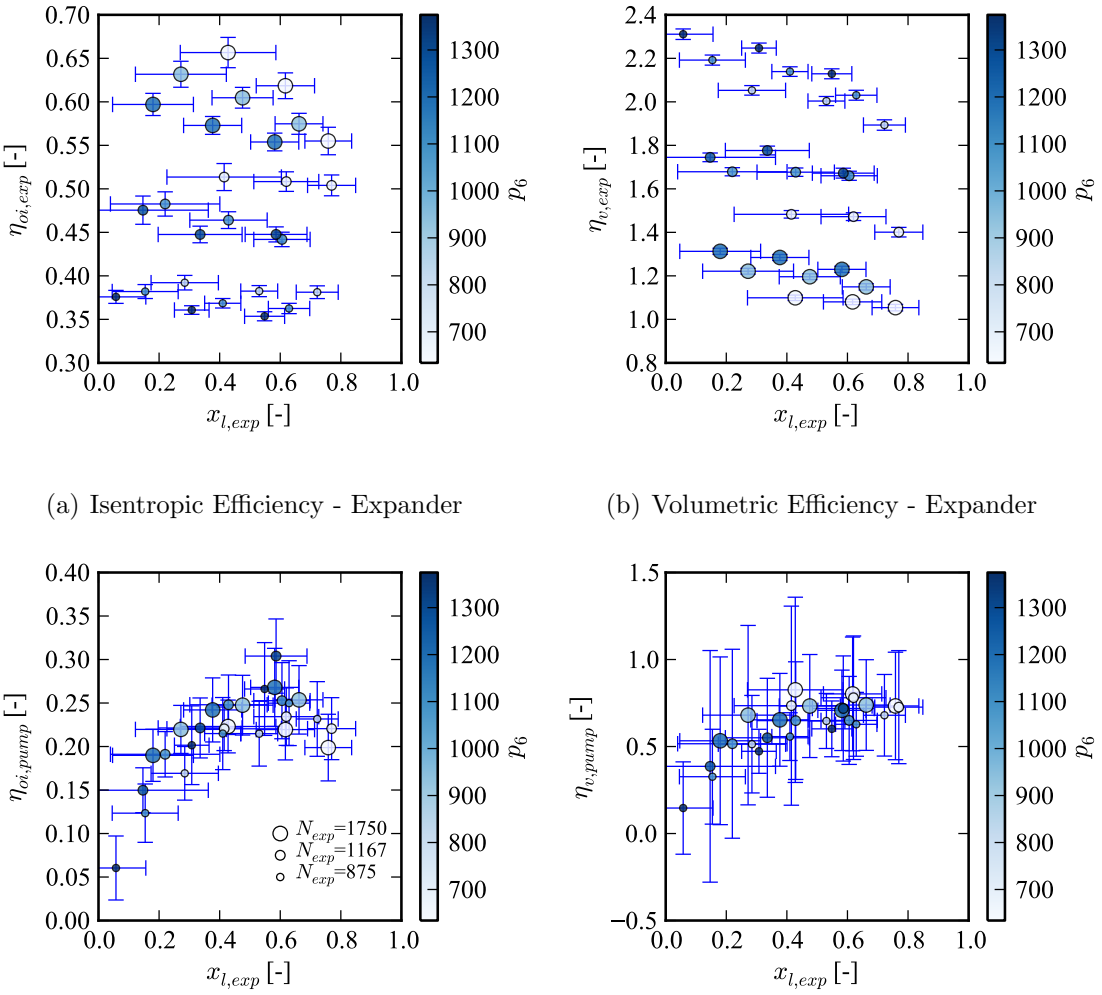
(b) Volumetric Efficiency - Compressor

(c) Isentropic Efficiency - Hydraulic Expander

(d) Volumetric Efficiency - Hydraulic Expander

Figure 6.13. Efficiencies of hot side rotating machinery components as a function of compressor oil mass fraction and compressor inlet pressure (marker color).

increases, the overall isentropic efficiency monotonically decreases. The same basic trend can be seen for the compressor volumetric efficiency, though the trend is not as clear and the decrease in volumetric efficiency is much less severe than the decrease in overall isentropic efficiency with oil flooding. The hydraulic expander has a poor isentropic efficiency, which in general increases with the suction pressure of the compressor (effectively equal to the discharge pressure of the hydraulic expander). The



(a) Isentropic Efficiency - Expander

(b) Volumetric Efficiency - Expander

(c) Isentropic Efficiency - Hydraulic Pump

(d) Volumetric Efficiency - Hydraulic Pump

Figure 6.14. Efficiencies of cold side rotating machinery as a function of expander oil mass fraction, expander rotational speed (marker size, where N_{exp} is rotational speed of expander in RPM) and expander inlet pressure (marker color).

volumetric efficiency of the hydraulic expander improved at higher oil mass fractions which correspond to higher rotational speeds of the hydraulic expander. In general leakage was a fairly significant factor for the hydraulic expander performance.

On the cold side of the rig, it is much more difficult to measure the oil mass fraction due to the small temperature differences over the components, and as a result, the

experimental uncertainty in the oil mass fraction is very large. Figure 6.14 presents the efficiencies of the expander and the hydraulic pump. For the expander, the efficiency is strongly driven by the rotational speed, and both volumetric efficiency and overall isentropic efficiency improve at higher rotational speed. This is due to a decrease in leakage at higher speed. In the limit that the rotational speed of the expander is zero, it behaves like a throttling valve that is entirely dominated by leakage. For each rotational speed, the expander overall isentropic efficiency tends to decrease with more oil flow, but the volumetric efficiency tends to improve with increased oil flow. The hydraulic pump has a maximum overall isentropic efficiency at an expander oil mass fraction of approximately 0.6, but only achieves an overall isentropic efficiency of 30%. The volumetric efficiency of the hydraulic pump tends to improve with the mass flow rate of oil, but the poor volumetric efficiency at lower oil mass fractions could be due to bubble entrainment rather than leakage or poor mechanical efficiency. Using the homogeneous void fraction model and a characteristic state point of 280 K, 400 kPa at the inlet to the hydraulic pump, an oil mass fraction of 98% (2% gas) corresponds to a gas void fraction of nearly 80%, as seen in Figure 6.15. Thus if there is even a small amount of gas entrained in the oil flow, the hydraulic pump no longer behaves as a volumetric oil metering device in which the oil flow rate is proportional to rotational speed. Bubbles of gas were visible to the naked eye, so a significant percentage of the hydraulic pump's displacement volume must have been taken up with gas, which can go some way towards explaining the discrepancy in flow rate predictions seen in Figure 6.9.

One of the causes of poor performance of the LFEC was the large pressure drops through the rig. Figure 6.16 shows the pressure drops through the high- and low-sides of the rig. The pressure drops are measured as the flange-to-flange pressure differences between the outlet of the compressor and the inlet of the expander and the outlet of the expander and the inlet of the compressor. In an ideal LFEC cycle, these pressure drops are zero, but in practice they were very large for the LFEC system constructed. On the high pressure side of the system, pressure drops over

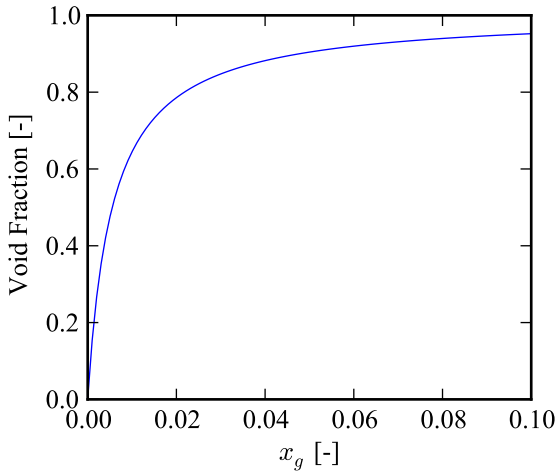
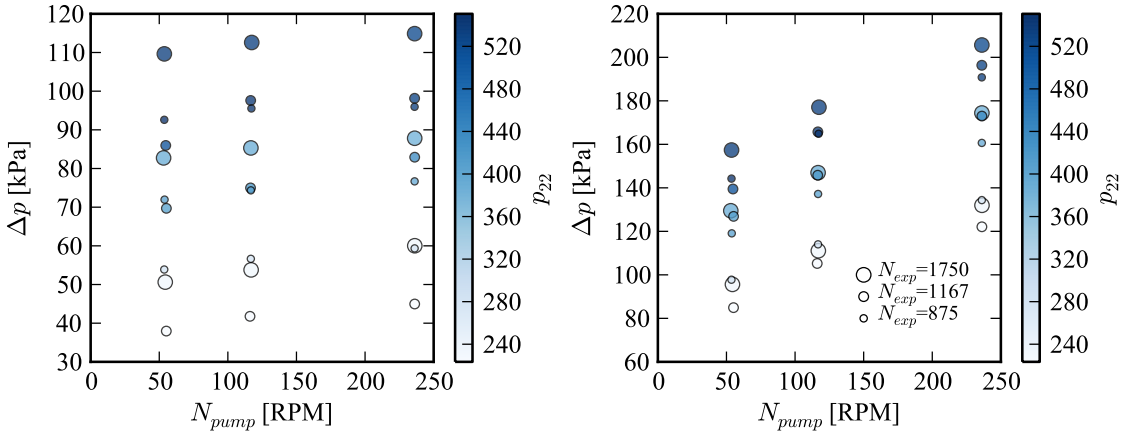


Figure 6.15. Void fraction at the hydraulic pump inlet as a function of entrained bubble gas mass fraction (T=280 K, p=400 kPa).



(a) High pressure side

(b) Low pressure side

Figure 6.16. Pressure drop through the high pressure side and low pressure side of the rig as a function of pump rotational speed, expander rotational speed (marker size, where N is rotational speed of expander in RPM) and compressor inlet pressure (marker color).

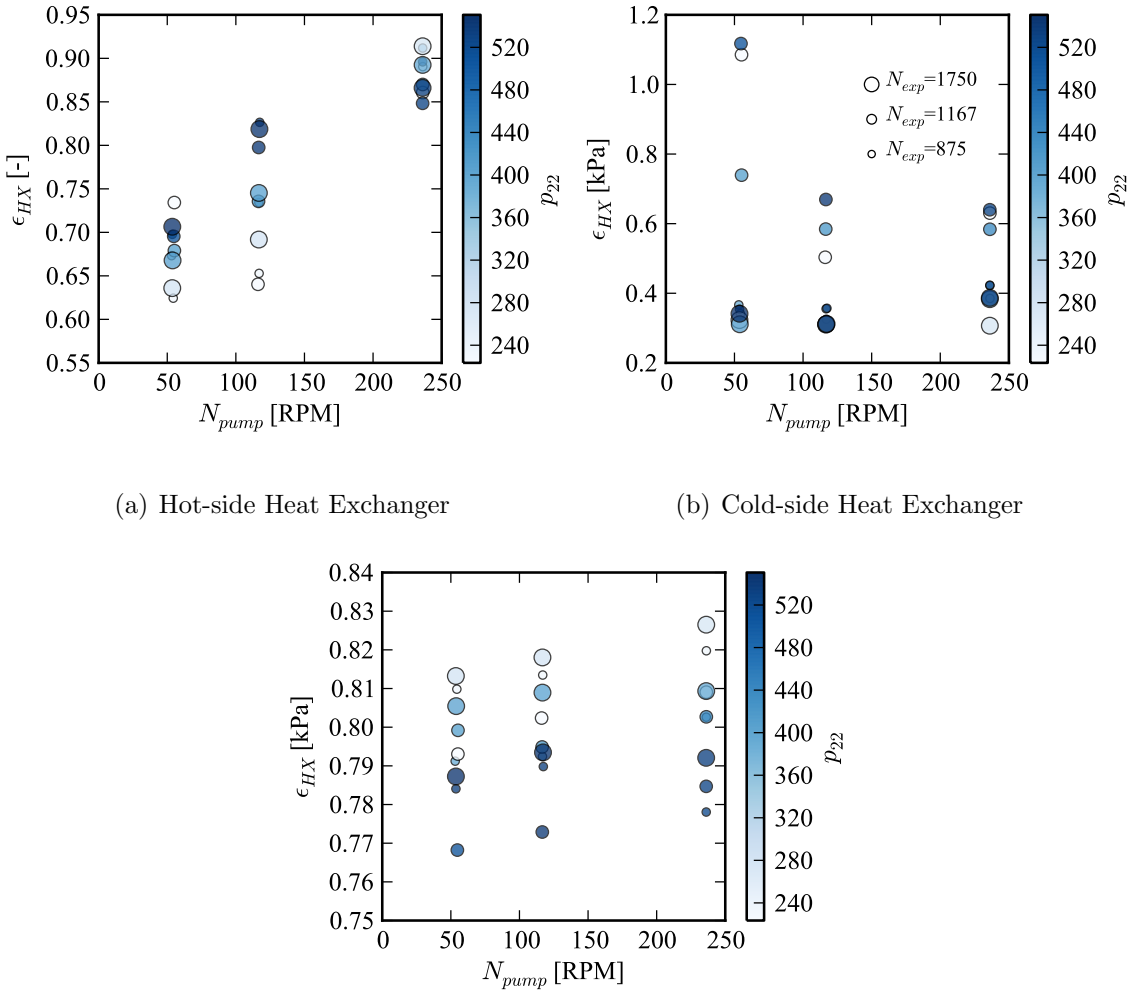
100 kPa are experienced, and on the low pressure side of the system, pressure drops as high as 200 kPa were experienced. On the low side of the system, a plurality of the pressure drop happened between the mixing point (point 21) and the inlet to the

compressor (point 22). The pressure drop over this section, comprising one hard 90° bend and a length of flexible hose was as much as 70 kPa. The design of systems for oil flooding requires a thorough understanding of the pressure drop of the flow of two-phase mixtures through bends, elbows and other flow elements. The same design rules-of-thumb that hold for single-phase flow are often not applicable for two-phase flow.

In addition to the poor performance of the rotating machinery and large pressure drops, the heat exchangers did not perform very well. Figure 6.17 shows the effectivenesses of the hot and cold heat exchangers and the regenerator, for which it is shown that the addition of oil improves the effectiveness of the hot heat exchanger. The cold heat exchanger had a few effectiveness values over 1.0 which is impossible, which means that the predicted oil flow rates for these points were significantly in error. The effectiveness values were evaluated based on the temperatures measured right at the heat exchanger, so the effects of thermal non-equilibrium also come into play. The relative uncertainties of the effectiveness values are relatively large, ranging from 2.80% to 15.24% for the hot heat exchanger, from 8.9% to 46.4% for the cold heat exchanger, and from 3.8% to 9.1% for the regenerator. It is difficult to see clear trends in the heat exchanger effectivenesses, but the regenerator and hot-side heat exchanger seem to perform the best, followed by the cold heat exchanger.

6.2.1 Liquid-Flooded Ericsson Cycle Concluding Remarks

The application of the Liquid-Flooded Ericsson Cycle to near-ambient temperature cooling applications does not appear to have any potential for competitive performance, particularly with the components that were used in the experimental testing. From the standpoint of model validation for the scroll compressor (presented here) and scroll expander (Lemort, 2008), the system performed adequately, but the use of larger oil separators would have been greatly beneficial. This would have allowed



(a) Hot-side Heat Exchanger

(b) Cold-side Heat Exchanger

(c) Regenerator

Figure 6.17. Effectiveness of the hot and cold LFEC heat exchangers as a function of pump rotational speed, expander rotational speed (marker size, where N is rotational speed of expander in RPM) and compressor inlet pressure (marker color).

for a direct measurement of the oil flow rate rather than an indirect measurement like the energy balance carried out here to back out the oil flow rate.

6.3 Scroll Compressor Model Tuning And Validation

Tuning of the liquid flooded scroll compressor model is carried out in a two-step process. First the mass flow rate is tuned based on leakage and pressure drop parameters, and then the shaft power is tuned based on mechanical loss, discharge pressure drop and external heat transfer parameters.

6.3.1 Mass Flow Tuning

A simultaneous optimization of inlet area factor $X_{d,inlet}$ and leakage gap widths was carried out in order to minimize the error in total mass flow. To carry out the optimization and minimize the number of optimization parameters, the flank leakage gap width was imposed to be equal to the radial gap width. In practice the radial and flank leakage gaps should be relatively similar. Therefore, the two independent parameters in the mass flow tuning are the fictional area fraction $X_{d,inlet}$ and the leakage gap width δ. During compressor operation, gas leaks from the higher pressure chambers to the lower pressure chambers. Since the compressor operates at a uniform rotational speed of 3500 RPM, the compressor leakage gap widths are assumed to be constant.

Figure 6.18 shows the mean absolute error for the prediction of the total mixture mass flow rate flowing through the compressor over the 27 data points selected for validation. The mean absolute error of N samples is defined by

$$MAE = \frac{\sum_{i=1..N} \left| \frac{\dot{m}_{model,i}}{\dot{m}_{exper,i}} - 1 \right|}{N}. \tag{6.15}$$

A gap width of approximately 15.43 μm and an inlet area correction factor of 0.417 give the best agreement with the experimental data obtained when using the corrected isentropic nozzle model. This optimal value was found by interpolating the data using quadratic interpolation and finding the minimum of the mean absolute error surface. To develop this figure, the values of δ employed were 10, 12, 14, 16 μm and values of $X_{d,inlet}$ of 0.3, 0.4, 0.5, and 0.6. Other models have been considered

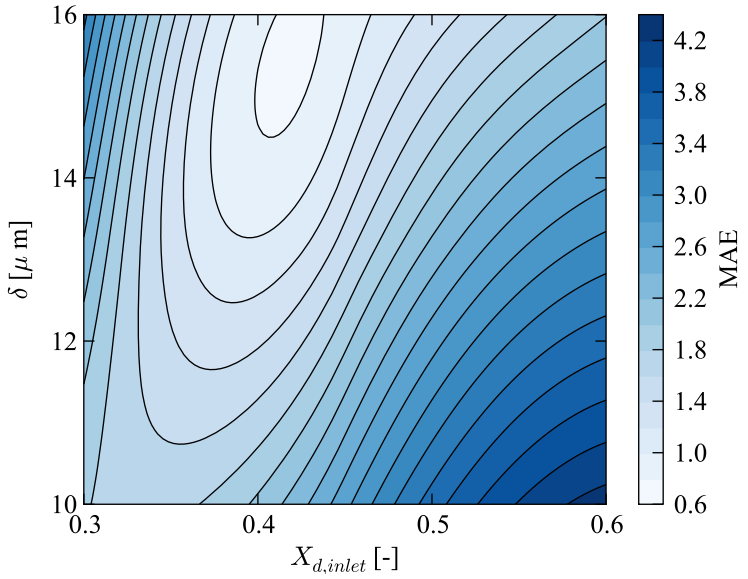


Figure 6.18. Mean absolute error of scroll compressor model mixture mass flow rate prediction for 27 validation state points as a function of δ and $X_{d,inlet}$ using the corrected isentropic nozzle leakage model.

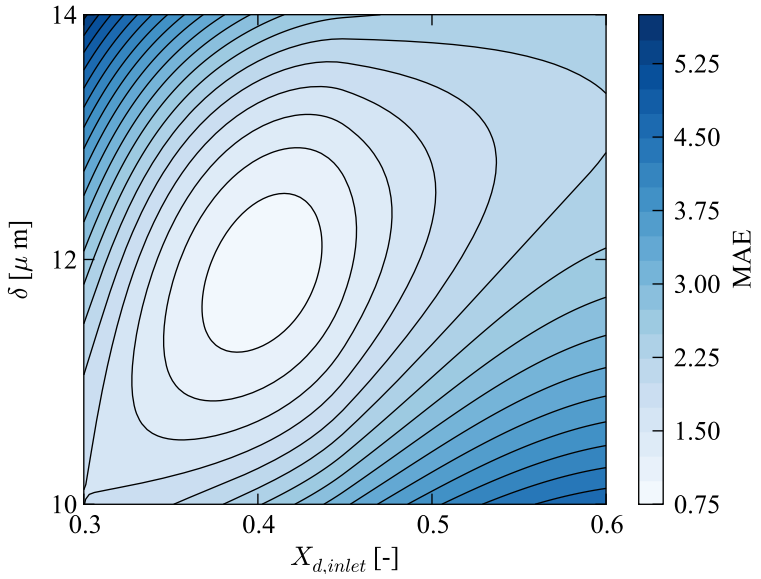


Figure 6.19. Mean absolute error of scroll compressor model mixture mass flow rate prediction for 27 validation state points as a function of δ and $X_{d,inlet}$ using the incompressible frictional laminar flow leakage model.

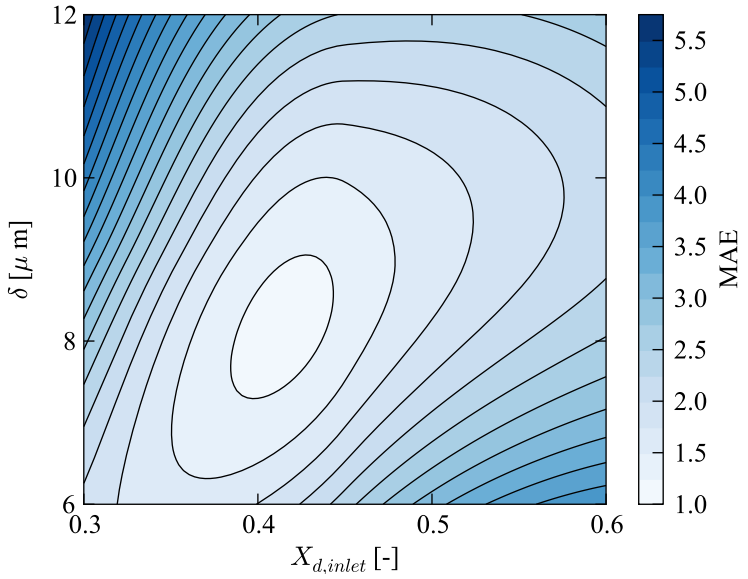


Figure 6.20. Mean absolute error of scroll compressor model mixture mass flow rate prediction for 27 validation state points as a function of δ and $X_{d,inlet}$ using the two-phase nozzle model for the leakage paths.

for the radial and flank leakage paths, including using laminar frictional flow (Bell et al., 2011), and the two-phase nozzle model. The same simultaneous optimization of gap width and inlet area factor can be carried out using these models. Figure 6.19 shows the results using the laminar frictional flow of gas through the leakage gap widths, for which an optimal gap width/inlet area fraction is approximately 12μm and 0.4 respectively, but the mean absolute error increases to 0.84%. Using the two phase nozzle model presented above for the other flow paths for the leakages yields an optimal gap width of approximately 8 μm and an inlet area fraction of 0.4 for an optimal mean absolute error of 1.18%, as seen in Figure 6.20. All three of the flow models yield the same inlet area fraction, and also yield relatively similar leakage gap widths. The optimized gap width for the two-phase nozzle is lower because of the flow of oil through the gaps yields a higher mass flux, and to yield the same leakage irreversibilities, the gap widths must be smaller.

It is quite difficult to say what the correct model for the flow through the leakages is. All the leakage models give similar predictions of the optimal gap width, but

ultimately the model (corrected isentropic nozzle) that yields the lowest mean average error in mass flow prediction seems like a reasonable choice.

To obtain the mass flow tuning results, a guess value for the mechanical losses was employed so that approximately the correct amount of heat transfer passes into the mixture in the suction chamber. For flooded compressors, the mass flow rate predictions are not sensitive to the heat transfer to the suction gas. For a representative state point from the experimental testing (T_s=317.0 K, p_s=421.8 kPa, p_d=1177.4 kPa, x_l=0.739), doubling the mechanical losses only decreases the mixture mass flow rate by 0.21 %. This shows that the prediction of the mixture mass flow rate is not very sensitive to the mechanical losses (and the heat transfer that takes the heat to the suction chamber).

6.3.2 Mechanical Losses Tuning

After the mass flow rates have been tuned, the next step is to tune the discharge port correction term and the mechanical losses. When no correction is made to the $X_{d,discharge}$ for the flow through the discharge port ($X_{d,discharge} = 1.0$), the mechanical losses are dependent on the oil-mass-fraction. From a consideration of the geometry and construction of the scroll compressor, it seems unlikely that the mechanical losses would be a strong function of the oil mass fraction unless other two-phase-flow phenomena not considered here were determined to be important. As a consequence, the discharge port discharge coefficient $X_{d,discharge}$ was tuned to remove the oil-mass-fraction dependence on the mechanical losses. Figure 6.21 presents the results for varying both the discharge port area correction factor as well as the mechanical losses. The values for $X_{d,discharge}$ employed were 0.3, 0.4, 0.5, and 0.6. The mechanical loss term $\dot{W}_{ML}$ was given the values 0.30, 0.35, 0.40, and 0.45 kW.

In general, decreasing $X_{d,discharge}$ tends to decrease the mechanical losses required in the model as the decrease in discharge port area results in increased flow irreversibilities for the discharge port flow. Based on an interpolated surface, the mini-

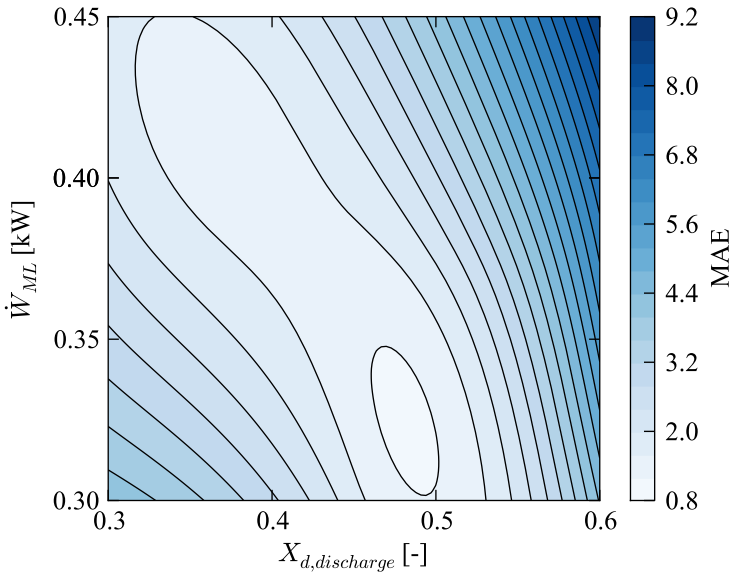


Figure 6.21. Mean absolute error of scroll compressor shaft power prediction for 27 validation state points as a function of $\dot{W}_{ML}$ and $X_{d,discharge}$.

mum MAE is obtained for the values $X_{d,discharge}$=0.482 and $\dot{W}_{ML}$ = 0.324 kW, but the lowest MAE for the prediction of the shaft power was obtained at the values of $X_{d,discharge}$=0.5 and $\dot{W}_{ML}$ = 0.4 kW. The constant mechanical loss torque model effectively captures the mechanical losses. Mechanical efficiencies ranged from 83% to 91%. A gap width of 15.43 μm and inlet area correction term $X_{d,inlet}$=0.417 were used with the corrected nozzle model for leakage.

6.4 Results Of Tuning Process

Finally the predictions of shaft power and mixture mass flow rate of the model can be compared together against the experimental data based on the optimal parameters obtained from the mass flow and mechanical loss tuning processes. The scroll compressor model successfully predicts the mixture flow rate through the compressor as well as the compressor shaft power with mean absolute errors for the mixture mass flow rate of 0.762% and 1.36% for the shaft power. Figure 6.22 shows the results

of the final analysis. All 27 points are predicted within an absolute error band of 3.4%. The mean uncertainty in the experimentally measured total compressor flow rate is 7.21%, and the mean uncertainty of the experimentally measured shaft power is 0.48%. The uncertainty in the oil mass fraction (an input to the numerical model) contributes to the error in predictions of the shaft power and the total mass flow rate.

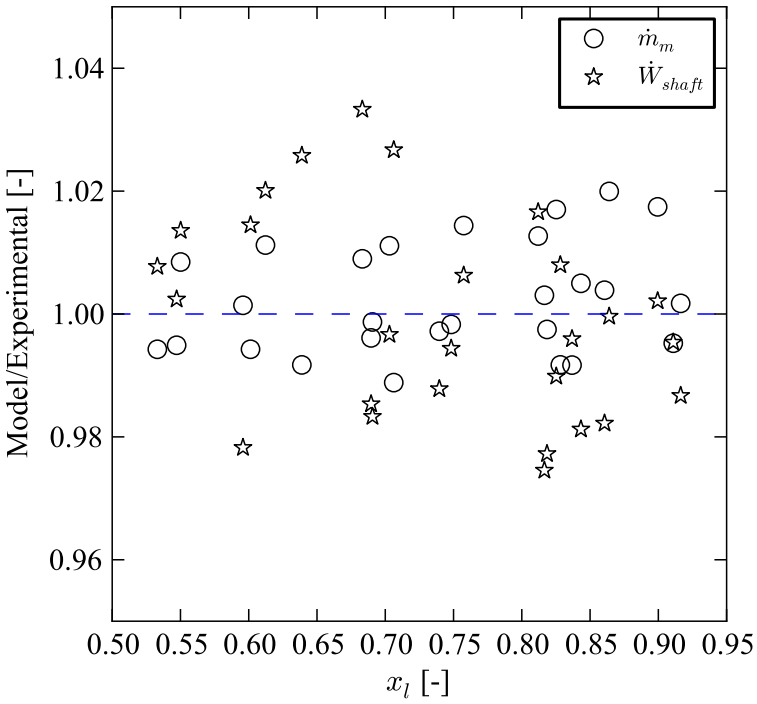


Figure 6.22. Compressor parity plot for predictions of the shaft power and the total mixture mass flow rate.

CHAPTER 7. OPTIMIZATION OF COMPRESSOR FOR LIQUID FLOODING

7.1 Motivation

While many of the compressor design rules-of-thumb are similar between liquid-flooded and conventional compressors, liquid-flooding fundamentally alters the thermodynamics of the compression process. As a result, the design of the compressor with liquid-flooding must also be altered to achieve the optimal performance. The goal of this chapter is to develop a formalized methodology that can be used to optimize the scroll wrap geometry of the scroll compressor with liquid flooding.

7.2 Optimization

The three applications for which a liquid-flooded compressor is optimized for are:

- The optimal operation point from the Liquid-Flooded Ericsson Cycle analysis identified by Hugenroth
- Transcritical CO_2 air-conditioning application
- A low-source-temperature R410A air-to-air heat pump for application in cold environments

There are a number of basic principles of design of compressors with liquid flooding. Firstly, it is necessary to avoid abrupt changes in direction of the liquid stream, particularly when the liquid is mixed with the working fluid. Changes in direction of the flow results in a large pressure drop, and this pressure drop contributes to the irreversibility of the compression process. Secondly, the suction and discharge ports

must be oversized in order to avoid large pressure drops. Thirdly, the built-in volume ratio of the compressor must be increased in order to better match the working process of the flooded compressor.

To begin the analysis, the ideal volume ratio for liquid-flooding is derived, as well as the optimal geometry of the involutes of the compressor with liquid flooding.

7.3 Derivation Of Ideal Volume Ratio For Liquid Flooding

The volume ratios needed for liquid-flooded scroll compressors are in general greater than those without liquid flooding. In order to derive the volume ratio required for the liquid-flooded compression process, the following constraints are imposed:

- Adiabatic efficiency of compression process from state point 1 to state point 2 given by η_a
- Initial and final masses are equal
- Initial and final oil mass fractions are equal
- Homogeneous mixture properties are employed
- Inlet condition (T_1, p_1, x_l) and discharge pressure p_2 imposed

Since the inlet state is known, the mixture properties at the inlet (h_1, ρ_1, s_1) can be obtained from the mixture model. The isentropic outlet temperature can be obtained from the equation

$$s_m(T_{2s}, p_2, x_l) = s_1 \tag{7.1}$$

where T_{2s} is iteratively determined using a numerical solver. The isentropic enthalpy is obtained from

$$h_{2s} = h_m(T_{2s}, p_2, x_l) \tag{7.2}$$

and based on the definition of the adiabatic efficiency, the outlet enthalpy is obtained from

$$h_2 = \frac{h_{2s} - h_1}{\eta_a} + h_1 \tag{7.3}$$

which allows solution for the actual discharge temperature T_2 from

$$h_m(T_2, p_2, x_l) = h_2 \tag{7.4}$$

using a numerical solver. With T_2 known, the density at state point 2 (ρ_2) can be obtained.

Conservation of mass for the compression process can be expressed as

$$\rho_1 V_1 = \rho_2 V_2 \tag{7.5}$$

and thus, if V_{ratio} is defined by $V_{ratio} = V_1/V_2$, then V_{ratio} can be given by

$$V_{ratio} = \frac{\rho_2}{\rho_1} \tag{7.6}$$

which yields the prediction for the optimal built-in volume ratio.

Figure 7.1 shows the results of the analysis presented in this section for the optimal point of the Liquid-Flooded Ericsson Cycle, which will be described further below. For reversible compression (η_a=1), the required volume ratio increases as the oil mass fraction increases, but at high oil mass fractions, the volume ratio goes to unity. When there is no oil-flooding (x_l=0), the ideal volume ratio can be found from the compression of pure gas. In the limit that the oil mass fraction goes to 1, the volume ratio does as well. This is due to the fact that for an incompressible fluid, an infinitely small decrease in volume is required to obtain an infinitely large increase in pressure.

When irreversibilities are added to the simple volume-ratio model by decreasing the adiabatic efficiency, the ideal volume ratio decreases. Irreversibilities in this simple model can be thought of as a heat source that is heating the mixture at constant volume. This model can accurately capture the effects of irreversibilities that manifest themselves in this fashion, which in practice is only the mechanical losses, which ultimately are transferred to the oil-gas mixture by heat transfer from the scrolls. For the same decrease in adiabatic efficiency, the flow irreversibilities (leakage and pressure drop) result in much greater decreases in the optimal volume ratio than predicted by the simple model. Leakage can be thought of as providing a resistance

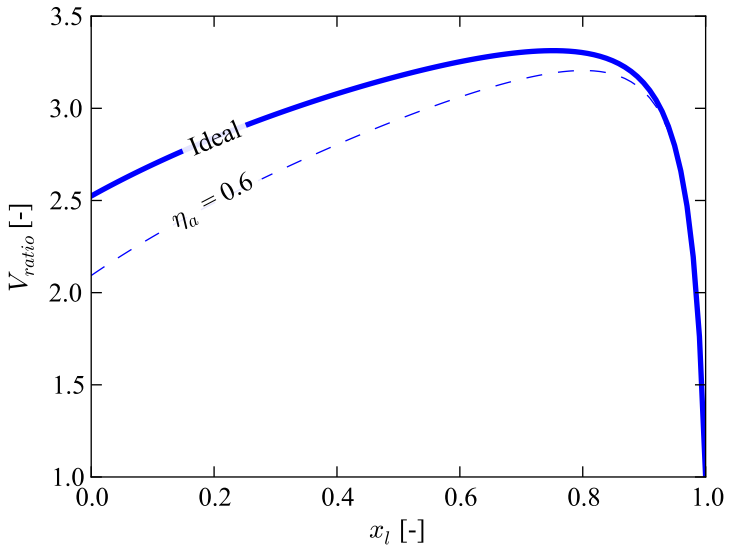


Figure 7.1. Predictions of optimal volume ratio from simplified model as a function of oil mass fraction (Nitrogen, Zerol, p_1=500 kPa, p_2=1850 kPa, T_1=278 K).

to increase in pressure of the gas-liquid mixure in the compression chambers. As a result, larger volume ratios are required to achieve the same increase in pressure since as the volume ratio is decreased, some of the boundary work is used to leak fluid back to lower pressure chambers.

The simple model presented here is primarily useful for getting a first-guess for the volume ratio required for the compression of a mixture of oil and gas - further optimization is required to obtain the best volume ratio of the compressor for a given application.

7.4 Definition Of Geometric Parameters

In general, the displacement of the compressor is governed by the capacity of the system that the compressor is being designed for. If the volume ratio and displacement of the compressor is then known, it is then possible to determine the scroll compressor geometry in order to match the desired volume ratio and displacement. A number of

different sets of constraints on the scroll geometry are possible, but the constraints employed here are that the volume ratio, displacement, and the thickness of the scroll wrap are fixed.

The displacement of the compressor is given by

$$V_{disp} = -2\pi h_s r_b r_o \left(3\pi - 2\phi_{ie} + \phi_{i0} + \phi_{o0}\right) \tag{7.7}$$

and the volume ratio of the compressor is given by:

$$V_{ratio} = \frac{V_{disp}}{2V_{c,d}} = \frac{3\pi - 2\phi_{ie} + \phi_{i0} + \phi_{o0}}{-2\phi_{os} - 3\pi + \phi_{i0} + \phi_{o0}} \tag{7.8}$$

where $V_{c,d}$ is the volume of one of the innermost compression chambers at the discharge angle. The thickness of the scroll wrap is given by

$$t_s = r_b(\phi_{i0} - \phi_{o0}) \tag{7.9}$$

and the orbiting radius is given by

$$r_o = r_b\pi - t_s \tag{7.10}$$

and if the displacement V_{disp}, volume ratio V_{ratio} and thickness of the scroll wraps t_s are imposed, then there are three equations (7.7, 7.8, 7.9) and 6 unknowns (ϕ_{ie}, ϕ_{i0}, ϕ_{o0}, ϕ_{os}, h_s, r_b).

Two further constraints are needed on the scroll geometry in order to fix the rest of the scroll geometry. Either ϕ_{o0} or ϕ_{i0} is a free variable, the other being fixed by the scroll wrap thickness for a given base circle radius. Increasing the value of ϕ_{i0} just rotates the scroll wrap, so for simplicity, ϕ_{i0} is set to zero. The value of ϕ_{os} is set to 0.3 radians.

Therefore, with the additional constraints imposed here, there remains just one free variable, which can either be taken to be the scroll wrap height h_s or the base circle radius r_b, and here the base circle radius was taken as the free variable. A method is presented in the next section to optimize the selection of the base circle radius.

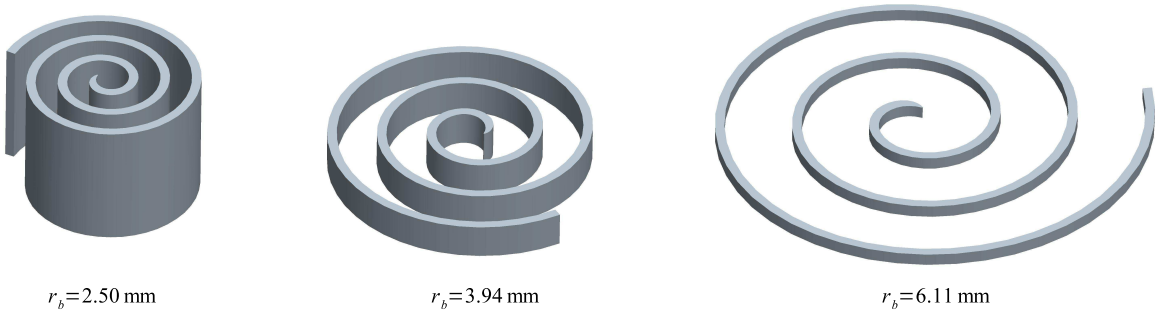


Figure 7.2. Family of scroll wraps for a volume ratio of 2.7, displacement of 104.8 cm^3, and wrap thickness of 4.66 mm.

With these constraints, it is possible to obtain an analytic solution for the relevant scroll wrap parameters. The outer involute initial angle is then given by

$$\phi_{o0} = -t_s/r_b \tag{7.11}$$

and after some algebra and simplification, the height of the scroll wrap is given by

$$h_s = \frac{V_{disp}}{2\pi r_b^2 V_{ratio}(\pi + \phi_{o0})(2\phi_{os} + 3\pi - \phi_{o0})} \tag{7.12}$$

and the ending angle of the scroll is given by

$$\phi_{ie} = \frac{V_{disp}}{4\pi h_s r_b^2(\pi + \phi_{o0})} + \frac{3\pi + \phi_{o0}}{2} \tag{7.13}$$

where both the fixed and orbiting scrolls have the same ending angle. If another set of constraints is desired, it is possible to use a non-linear solver like that available in Engineering Equation Solver (EES) to obtain the scroll wrap geometry.

For the same volume ratio and displacement, the larger r_b is, the smaller h_s must be to maintain the same displacement and scroll wrap thickness. This yields a family of solutions from a very narrow cylinder to a "pancake" scroll design. Selected members of this family are shown in Figure 7.2. All scroll wraps are plotted at the same scale.

7.5 Derivation Of Optimal Base Circle Radius

As shown in the above section, for a given volume ratio, displacement, and scroll wrap thickness, a family of different scroll wraps can be obtained. The range of scroll

wraps, from a narrow cylinder to a pancake scroll, offer different performance due to the variation in the leakage rates. It is therefore useful to develop a simple model for the leakage terms in order get a first guess for the optimal scroll wrap geometry from a leakage standpoint. In the above analysis, the base circle radius r_b was a free variable, but the model presented here can predict the optimal base circle radius with reasonable accuracy.

To begin the analysis, it is first assumed that some portion of the scroll wrap does not contribute to radial leakage. This can be understood by considering the suction chamber. Over the course of the first rotation, the outermost conjugate point moves 2π radians towards the center of the compressor. Radial area between the suction chamber and the suction area does not contribute to leakage since there is effectively no pressure difference to drive the flow. Therefore, an effective ending angle of the scroll wrap is defined by

$$\phi_{ie}^* = \phi_{ie} - \pi \tag{7.14}$$

which removes the contribution of half of the suction chamber since over the course of one rotation, the mean conjugate angle is the inner ending angle minus a half rotation or π radians. The same argument is employed for the inner starting angle in the discharge region. Once the discharge region has equalized in pressure the radial leakage area no longer contributes to leakage. Therefore in the discharge region, another π radians are removed from the scroll involute, yielding an effective inner involute starting angle of

$$\phi_{is}^* = \phi_{is} + \pi \tag{7.15}$$

which removes the contribution from the portion of the rotation where the discharge region is equalized in pressure. Thus the total radial leakage area based on the inner involutes of the fixed and orbiting scrolls can be given by

$$A_{radial}^* = 2\delta_{radial} \int_{\phi_{is}^*}^{\phi_{ie}^*} r_b(\phi - \phi_{i0}) d\phi \tag{7.16}$$

which yields

$$A_{radial}^* = 2r_b\delta_{radial} \left(\frac{(\phi_{ie}^*)^2}{2} - \frac{(\phi_{is}^*)^2}{2} \right) \tag{7.17}$$

because the inner initial angle ϕ_{i0} was fixed at 0 in order to derive the involute parameters.

The flank area is determined by the number of flank contact points in existence over the course of a rotation. The mean total number of flank contact points is given by

$$N_{flank} = 2\frac{\phi_{ie} - \phi_{is}}{2\pi} \tag{7.18}$$

and the flank leakage flow area for each contact point can be given by $h_s\delta_{flank}$, thus the total flank leakage area is given by

$$A^*_{flank} = F\delta_{flank}h_s N_{flank} \tag{7.19}$$

where F is a flow adjustment parameter. For a given flow area and pressure difference, more flow will go through the flank leakage. This can be understood by considering the hydraulic diameters of the leakage paths. In the radial leakage the hydraulic diameter is always twice the gap width, while for the flank leakage the conformal contact results in a hydraulic diameter that increases sharply away from the throat of the leakage path. The ratio of flank to radial frictional leakage mass fluxes is approximated from the mass flow correction terms, and is given by a value for F near 3. This value was slightly tuned in order to better fit the results from the optimization carried out on the Liquid-Flooded Ericsson Cycle compressor presented below for a volume ratio of 2.7. In practice, the value of this ratio is dependent on the thickness of the scroll wrap and the system operating parameters, but since the purpose of this section is to derive a guess value for detailed optimization, this value is sufficiently accurate. Thus the total effective leakage is given by

$$A^*_{total} = A^*_{radial} + A^*_{flank} \tag{7.20}$$

and the results for the effective leakage areas as a function of base circle radius for a volume ratio of 2.7 and displacement of 104.8 cm^3 are shown in Figure 7.3. The effective radial leakage increases linearly with the base circle radius, while the effective flank leakage decreases with the base circle radius. Thus the sum of the two terms yields a minimum effective leakage area at a base circle radius of 3.91 mm.

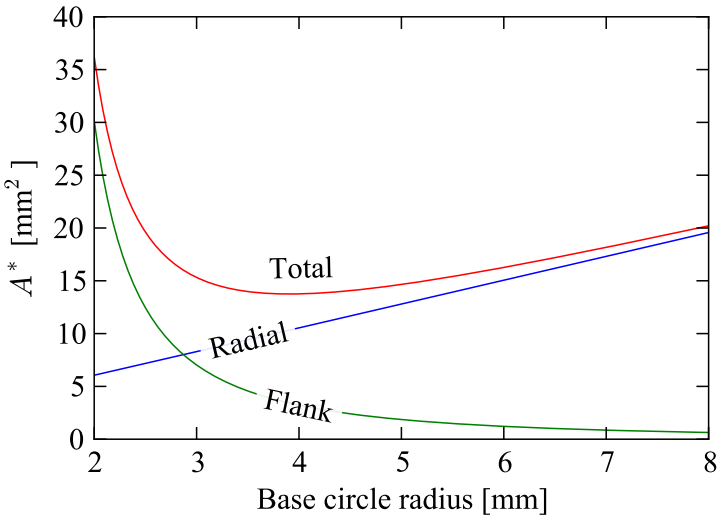


Figure 7.3. Effective flank and radial leakage areas for compressor with volume ratio of 2.7, displacement of 104.8 cm^3, scroll thickness of 4.66 mm.

Therefore it is clear that there is a base circle radius that optimizes the performance of the compressor by minimizing the effective leakage area. Thus a numerical optimization routine can be employed to determine the optimal base circle radius over a range of displacement and volume ratios for a fixed scroll wrap width of 4.66 mm. The results of this analysis are shown in Figure 7.4. The optimal base circle radii obtained from the detailed compressor modeling for the Liquid-Flooded Ericsson Cycle and the CO_2 analyses presented in the following sections are also overlaid in order to demonstrate the effectiveness of this method for calculating an approximate optimal base circle radius. It is straightforward to generate a similar plot for a different scroll wrap thickness. These results show that for a given displacement, as the volume ratio increases, the optimal base circle radius decreases. Furthermore, for a given volume ratio, as the displacement is increased, the optimal base circle radius increases. This chart can be generally employed in the design of scroll wraps, whether for flooded or dry compression applications. The inclusion of geometrically-dependent mechanical

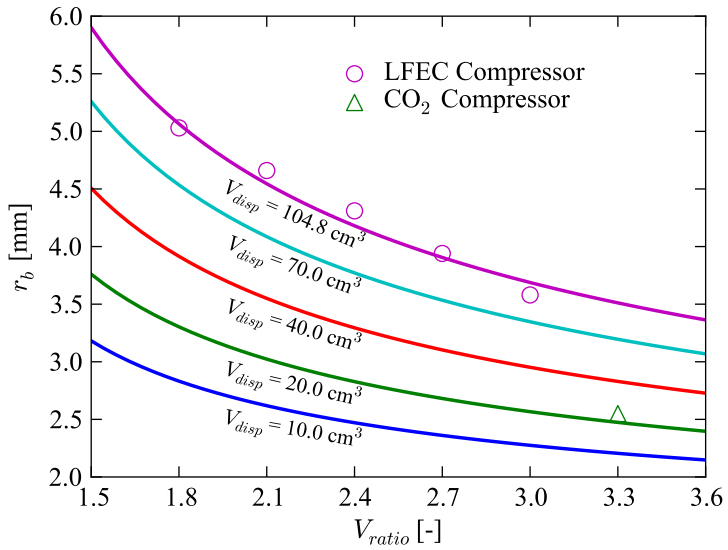


Figure 7.4. Optimal base circle radius as a function of displacement and volume ratio for a scroll with thickness 4.66 mm.

losses and scroll wrap manufacturing cost would result in a different optimal scroll wrap geometry.

7.6 Compressor Optimization For Ericsson Cycle Optimal Point

From the cycle analysis of Hugenroth et al. (2006) it was found that there exists an optimal operation point for the Liquid-Flooded Ericsson Cycle that yields the maximum cycle Coefficient of Performance (ratio of cooling capacity to net input power). In order to achieve the desired COP of 1.25, suction and discharge pressures of 500 kPa and 1850 kPa respectively are used with an assumed compressor adiabatic efficiency of 87%. This is a very challenging target for compressor design. In this section, a method is proposed for carrying out the optimization process to approach this target efficiency. While the optimization process of Hugenroth et al. (2006) was based on a smaller cooling capacity, here the suction displacement volume is fixed to be equal to that of the compressor used in the experimental testing of the Liquid-Flooded Ericsson Cycle. The optimization process begins with the use of the baseline

Sanden automotive compressor. The compressor inlet temperature is fixed at 5°C as the cool thermal reservoir for the LFEC is set at 2°C. For Zerol 60 (an alkylbenzene refrigeration oil) and nitrogen as the working fluid pair, Hugenroth found that the optimal ratio of oil capacitance rate to gas capacitance rate was 12.47, or an oil mass fraction of 0.8796. In spite of the large mass fraction of oil, at the inlet of the compressor the gas still occupies 95.14% by volume (based on homogeneous void fraction) because of the large difference in densities between the oil (865 kg/m^3) and the gas (6 kg/m^3).

7.6.1 Assumptions And Constraints

In order to carry out the optimization process, the following constraints were imposed on the compressor:

- Displacement of compressor held constant at 104.8 cm^3 (same as baseline compressor).
- Thickness of scroll wrap held constant at 4.66 mm (same as baseline compressor) in order to ensure that the scroll wraps are sufficiently stiff enough to handle the mechanical load.
- Inner scroll wrap initial angle held at 0.0.
- Inner scroll wrap starting angle given by $\phi_{is}=\phi_{i0}+\pi$. An offset of π radians between inner initial and inner starting angles yields a reasonably open discharge region which allows for a large discharge port
- Outer scroll wrap starting angle given by $\phi_{os}=\phi_{is}-\pi+0.3$.
- Mechanical efficiency of compressor held constant at 90%. In practice the mechanical efficiency of the scroll compressor will also be impacted by the particulars of the scroll geometry and its construction. 90% is believed to be a

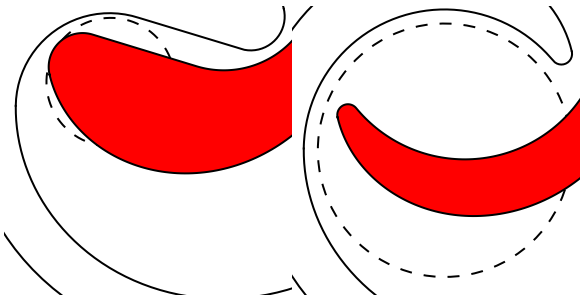


Figure 7.5. Discharge port blockages at $\theta = 7\pi/4$ for baseline compressor (left) and 2 arc discharge with larger discharge port (right).

reasonable estimate of the mechanical efficiency based on the data from the scroll compressor model tuning.

7.6.2 Discharge Geometry

The Sanden TRS-105 compressor has a quite small discharge port (12 mm diameter) for the large amount of oil flow that flows through it, so the first step considered to optimize the scroll machine is to give the discharge region a larger port of 0.9 times the radius of the arc which closes the involute curves, which yields a discharge port diameter of 24 mm. Another factor which negatively impacts the pressure drop from the discharge region is blockage of the discharge port which occurs due to the tip of the orbiting scroll.

First the baseline scroll machine is modified to use two arcs to close the involute pair at the discharge in order to decrease the amount of discharge port blockage which occurs over one rotation. In addition, the discharge port is increased in size to near the largest port size which will fit in the discharge region, as seen in Figure 7.5. The free discharge port area can be calculated using a numerical integration scheme, and can be seen in Figure 7.6.

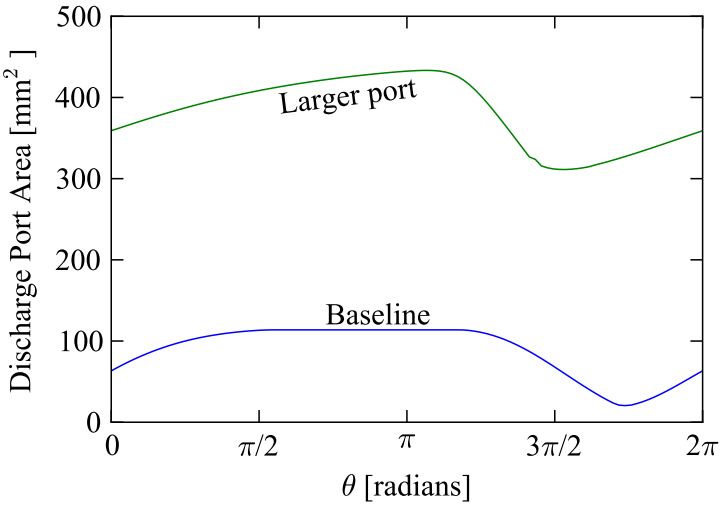


Figure 7.6. Discharge port free area over one rotation.

7.6.3 Suction Geometry

The flow entering the Sanden compressor must change direction in order to enter the suction pockets. In a practical compressor this problem can be remedied by increasing the diameter of the suction ports as well as splitting the flow into two streams, each of which inject directly into the suction pocket without a change in direction. In order to account for having two separate suction ports for the compressor, the area correction term which was included in the suction flow model to account for the pressure drop due to the bends is removed ($X_{d,inlet} = 1$). The pressure-crank angle plot for this system is shown in Figure 7.7. An overall isentropic efficiency of approximately 64.6% is predicted.

Clearly the suction and discharge port pressure drop components have been successfully controlled using the modifications presented here, though there are still two major irreversibilities remaining, the under-compression and leakage losses.

In order to decrease the under-compression losses, it is necessary to compress the gas-liquid mixture to nearer the discharge pressure, which requires a larger volume ratio. Figure 7.8 shows the results of varying the volume ratio from 1.8 to 3.0 and

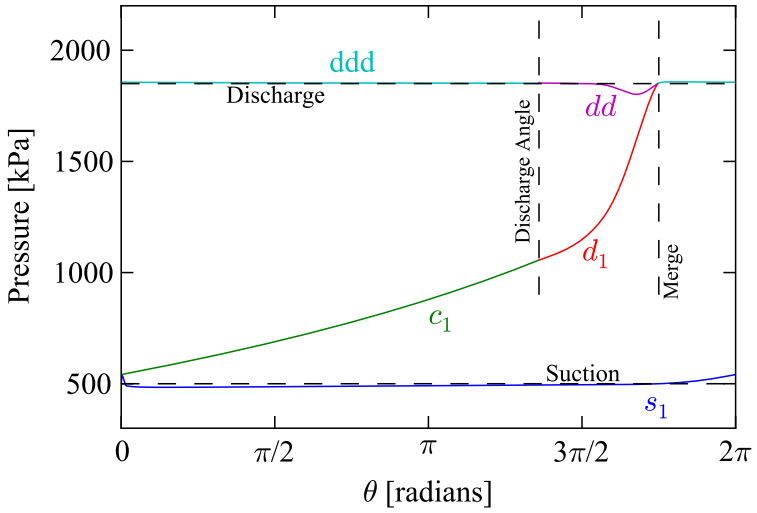


Figure 7.7. Pressure versus crank angle for two-inlet compressor with larger discharge port.

altering the base circle radius between 2 mm and 9 mm. For each volume ratio, there is an optimal base circle radius that maximizes the overall isentropic efficiency by minimizing the leakage losses. The optimal base circle radii are shown in Figure 7.4. Based on the simple modeling presented at the beginning of this chapter, for an adiabatic compression process, the ideal volume ratio should be around 3.1, but the optimal volume ratio obtained from the detailed modeling results presented here is 2.7. Even so, these results demonstrate that the simplified volume ratio model can adequately predict the ideal volume ratio.

The compressor can be further optimized by decreasing the leakage gap width and mechanical losses. In practice, decreasing the gap width can be quite difficult. With a gap width of 6 μm, the overall isentropic efficiency of the compressor is over 85%. The mechanical losses can also be decreased in the model, but in practice it is difficult to decrease the mechanical losses. The detailed modeling suggests that a target of 80% overall isentropic efficiency would be possible, but getting to 87% would not be feasible. To get to an 87% overall isentropic efficiency would require essentially no pressure drops, a very small amount of leakage, and a mechanical efficiency over 95%.

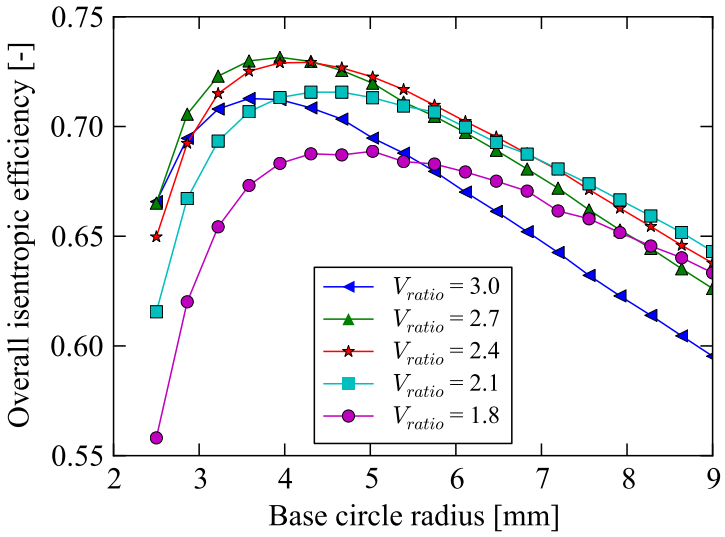


Figure 7.8. Overall isentropic efficiency for optimized compressor for Liquid-Flooded Ericsson Cycle Application.

7.7 Compressor Optimization For Liquid-Flooded CO_2 Air Conditioning

Another application that would appear promising for the application of liquid flooding is the compression of CO_2 in transcritical CO_2 air-conditioning systems. In this case, the displacement of the compressor is fixed at 20 cm^3, and the imposed pressures for the compressor are set at 3000 kPa (30 bar) suction pressure, and a discharge pressure of 10000 kPa (100 bar).

The same optimization procedure is employed for the CO_2 compressor. The steps are:

1. The displacement and imposed state points are given by the liquid-flooded cycle modeling
2. The initial guess for the volume ratio is given by the analysis in Section 7.3
3. The initial guess for the optimal base circle radius is given by the analysis in Section 7.5

Figure 7.9 shows the optimal volume ratios for CO_2 liquid-flooded compressors as a function of liquid mass fraction for Zerol and water as the flooding agents for the

state point under consideration here. Because water has a much higher specific heat than the Zerol alkyl-benzene oil, for the same liquid mass fraction, a larger volume ratio is required. This is because the water-CO_2 compression process is much closer to isothermal due to water's high specific heat. From a cycle performance standpoint, water is a far superior flooding agent due to its high density, low CO_2 solubility and high specific heat, but the necessity of a larger volume ratio (and commensurate increase in scroll wrap machining) means that water is not the ideal flooding agent from a compressor-design standpoint.

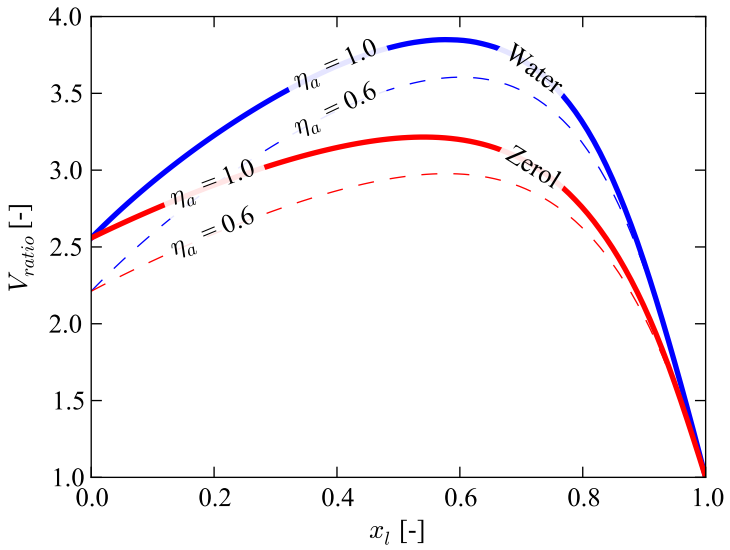


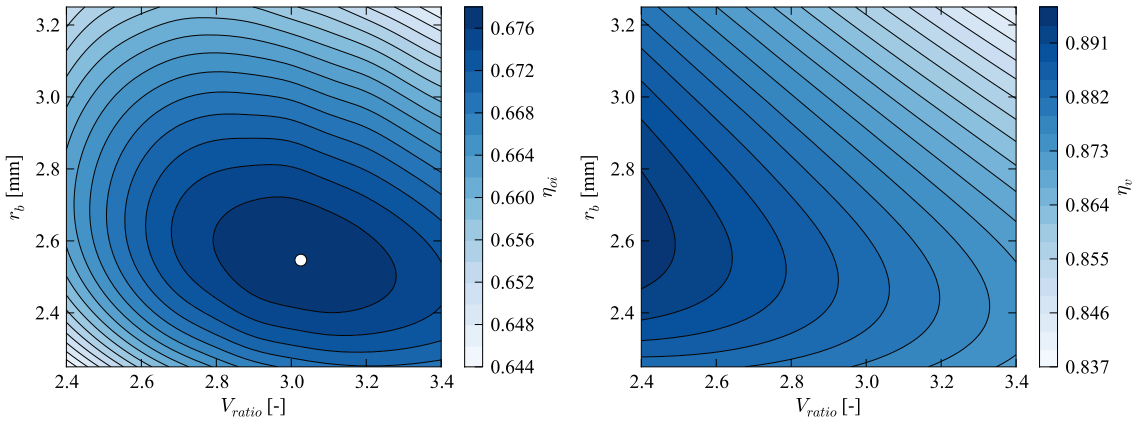
Figure 7.9. Predictions of optimal liquid-flooded volume ratio from simplified model as a function of liquid mass fraction (CO_2, p_1=3000 kPa, p_2=10,000 kPa, T_1=310 K).

The detailed scroll compressor model is then run for a range of volume ratios and base circle radii for a fixed leakage gap width of 12μm for both flank and radial leakages. The discharge port is set to be as large as permitted by the curves in the discharge region, and the inlet area into the scroll compressor is set to be equal to

$$A_{inlet} = 4h_s r_o \tag{7.21}$$

which assumes two rectangular symmetric ports, each with height equal to the scroll wraps and width equal to twice the orbiting radius. In general, the ports for CO_2 do not need to be as large since the density of CO_2 is much higher than that of other common refrigerants for the same operating point. A mechanical efficiency of 90% is assumed.

The results from the scroll compressor model are fit with polynomial regressions in order to obtain the optimal compressor geometry. The contour plots in Figure 7.10 show the results for both volumetric and overall isentropic efficiency of the compressor with 12 μm leakage gap widths. The surfaces are relatively flat, which suggests that there is quite a bit of flexibility in scroll wrap design. The geometry which optimizes the overall isentropic efficiency does not also optimize the volumetric efficiency. This is due to the fact that as the volume ratio is increased, there is more leakage, but there are less under-compression losses. As a result, smaller volume ratios result in less leakage irreversibilities and larger volume ratios result in less under-compression losses. In general, it is better to design the compressor to optimize the overall isentropic efficiency, then increase the displacement to get the desired flow rate.



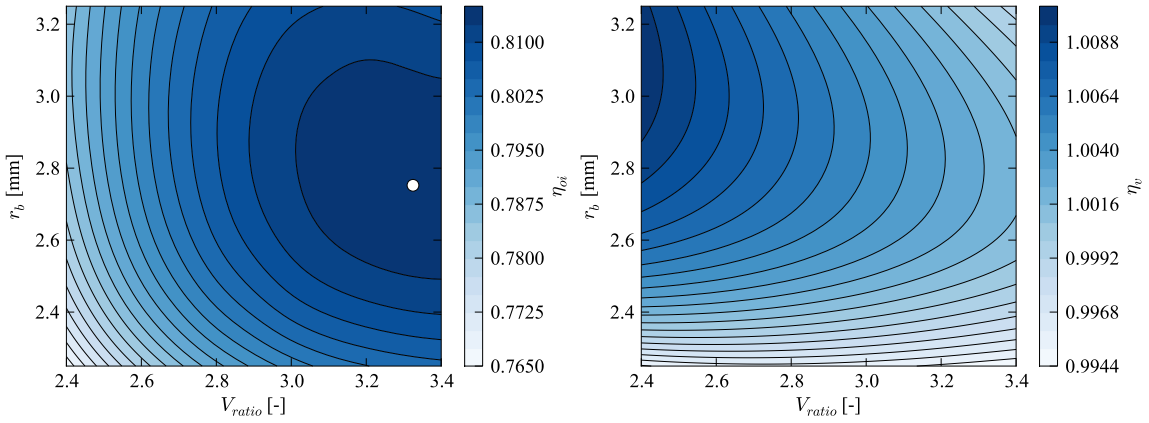
(a) Overall Isentropic Efficiency

(b) Volumetric Efficiency

Figure 7.10. CO_2 efficiency terms as a function of base circle radius and volume ratio for $\delta_{radial} = \delta_{flank} = 12\mu$m (○: optimal performance point).

With a 12 μm leakage gap width for both radial and flank gap widths, the maximum overall isentropic efficiency of 0.678 is reached at a base circle radius r_b of 2.55 mm and a V_{ratio} of 3.03. This base circle radius is slightly higher than the optimal base circle radius obtained for the Liquid-Flooded Ericsson Cycle application, though the imposed pressure difference over the compressor is 70 bar while it is only 13.7 bar for the LFEC compressor. As a result, there tends to be more leakage for CO_2 compressors due to their large pressure lift.

Figure 7.11 shows the results for smaller leakage gap widths. If the leakage gap widths are decreased to those of Ishii (2008) of 3μm radial gap width and 6μm flank gap width, the optimal overall isentropic efficiency of 0.815 is obtained for a base circle radius of 2.75 mm and V_{ratio} of 3.32. Both the overall isentropic and volumetric efficiencies are greatly improved, and the volumetric efficiency is above 1 in some cases. Volumetric efficiencies above 1 are possible due to the super-charging effect in the suction process. In addition, due to the decrease in leakage irreversibilities, the volume ratio increases, to nearer the ideal value predicted in Figure 7.9.



(a) Overall Isentropic Efficiency

(b) Volumetric Efficiency

Figure 7.11. CO_2 efficiency terms as a function of base circle radius and volume ratio for $\delta_{radial} = 3\mu$m and $\delta_{flank} = 6\mu$m ($\circ$: optimal performance point).

The results of the optimization for the CO_2 compressor suggest that an efficient water-flooded compressor could be constructed. The critical caveat of these results for the CO_2 compressor are that water is a very poor lubricant, and for that reason alone, it might be necessary to use an oil as the flooding agent. The system performance using water as the flooding agent does seem quite promising.

Appendix G contains the drawings for a transcritical CO_2 compressor for oil-flooded air conditioning applications. This compressor has a displacement of 20 cm^3 and volume ratio of 2.35. A smaller volume ratio was used for ease of manufacture.

7.8 Compressor Optimization For Liquid-Flooded R410A Air-Source Heat Pump

Another application for which liquid-flooding has promise is in low-ambient temperature air-source heat pumps. This application requires large temperature lifts for which the standard systems have poor efficiency.

In order to optimize the compressor for low-source-temperature heat pumps it is necessary to have a rating point to optimize the compressor for. In this case, the design condition is the same as employed by Bertsch (2005), or 17.6 kW of heating capacity at an air-source temperature of -10°C. Analysis of the experimental data from Bertsch suggests that an evaporating dew temperature of approximately -25°C could be expected. The condensing dew temperature for the cycle is assumed to be 43.3°C (110°F) in order to yield hotter return air temperatures from the heat pump condensing coil because of human comfort concerns. Based on the analysis of Chapter 3, the oil mass fraction can be swept over a range of values, and at each oil mass fraction, the system heating COP can be calculated, and the optimal system heating COP of 2.7 is obtained at an oil mass fraction of 0.557. At this oil mass fraction, a compressor displacement of 98 cm^3 is required, and the suction and discharge pressures for the compressor are 329.45 kPa and 2600.3 kPa respectively. The inlet temperature to the compressor is 35.25°C. This pressure ratio of 7.9 is a large pressure ratio for a scroll compressor, but with all the oil, the discharge

temperature is only 77.36°C, whereas over these conditions without flooding, the discharge temperature of the compressor would likely be above 120°C.

Detailed modeling of an off-the-shelf hermetic R410A compressor has given the effective gap widths of 15 μm and a motor-mechanical efficiency of 85%.

Based on a survey of rating data of Copeland scroll compressors for these conditions, it seems an overall isentropic efficiency (including the motor losses) of 66% could be expected for a conventional, off-the-shelf hermetic scroll compressor operating with the refrigerant R404A.

Figure 7.12 shows the optimal volume ratio for the R410A scroll compressor, determined using the simplified analysis with POE oil as the flooding agent. The volume ratio required for the R410A compressor is quite large compared with the optimizations carried out for the CO_2 and LFEC compressors because the imposed pressure ratio is much larger.

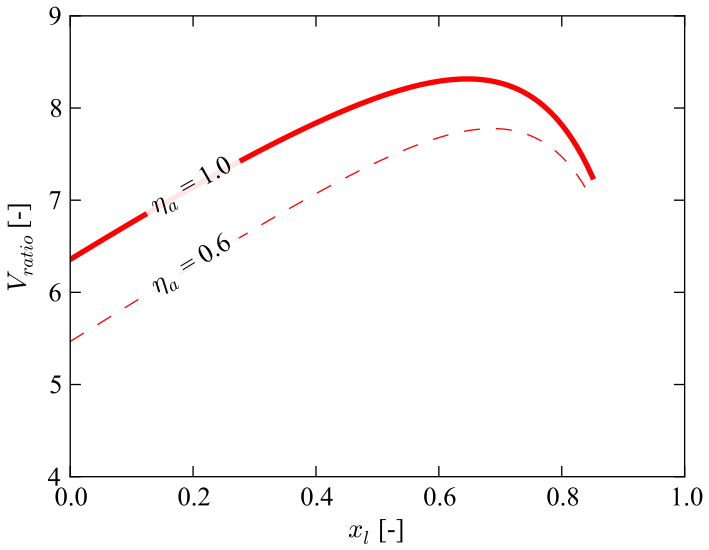


Figure 7.12. Predictions of optimal liquid-flooded volume ratio from simplified model as a function of oil mass fraction (R410A, POE oil, p_1=329.5 kPa, p_2=2600.3 kPa, T_1=35.3°C).

Finally the volume ratio of the scroll compressor is varied, and the performance of the scroll compressor with oil flooding can be predicted based on the detailed

compressor model. Figure 7.13 shows the overall isentropic efficiency of the scroll compressor as a function of the volume ratio. The base circle radius is imposed to be the optimal base circle radius from the above analysis. Even a volume ratio of 6 is below the optimal volume ratio. As the volume ratio is increased, the under-compression losses are decreased, but the leakage losses are increased due to the longer length of the scroll wrap. From a manufacturing standpoint, smaller volume ratios are easier to machine since there is less milling required to form the scroll wraps. Also, in practice discharge valves are commonly used to overcome under-compression losses. Discharge valves are beneficial if the discharge pressure drop introduced by the discharge valve creates less irreversibility than the under-compression losses without the discharge valve. Due to the large amount of oil flowing through the compressor, it was decided that discharge valves would not be an option due to mechanical stresses on the valve.

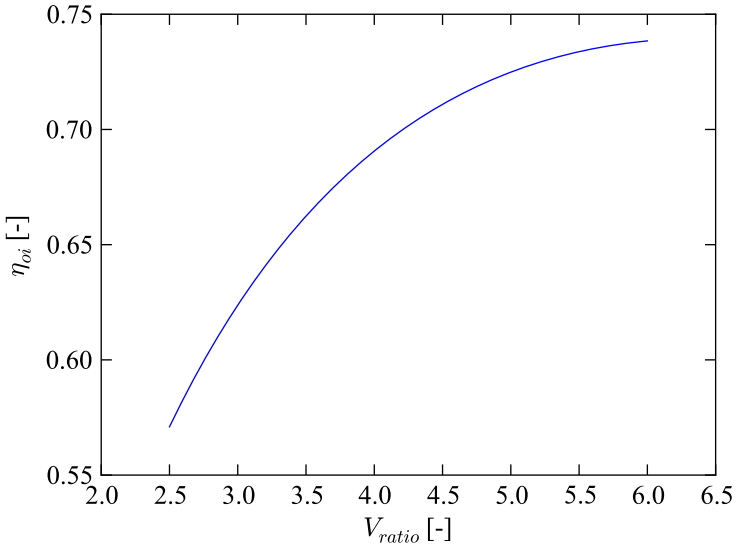


Figure 7.13. Model predictions of overall isentropic efficiency of R410A compressor with displacement of 98 cm^3, scroll wrap thickness of 3.0 mm and optimal base circle radius (R410A, POE oil, p_1=329.5 kPa, p_2=2600.3 kPa, T_1=35.3°C, x_l=0.557).

Figure 7.14 shows both the scroll wraps and the pressure-crank angle plots for a range of volume ratios spanning the volume ratios investigated for the R410A low-temperature heat pump. It is clear that as the volume ratio is increased, the pressure at the discharge angle is increased, decreasing the under-compression losses. Another clear problem with using constant-wall-thickness scroll wraps is that as the scroll wraps get longer and longer (large volume ratio), the effective derivative of the volume for the outer compression chambers decreases. For the volume ratio of 2.5, the derivative of the volume of the compression chambers with respect to the crank angle is -3.83x10^{-6} m^3/radian while for the volume ratio of 5.5, it is -1.67x10^{-6} m^3/radian. As a result, the outermost compression chambers do not contribute much to the increase in pressure. This can be seen by considering the pressure at the end of the first rotation of the outer-most compression chambers. For a volume ratio of 2.5, the pressure of the outermost compression chamber at the end of the first rotation is 723 kPa, while for a volume ratio of 5.5, it is 437 kPa. Based on the geometric analysis presented above, the derivative of the compression chamber volume with respect to the crank angle is given by

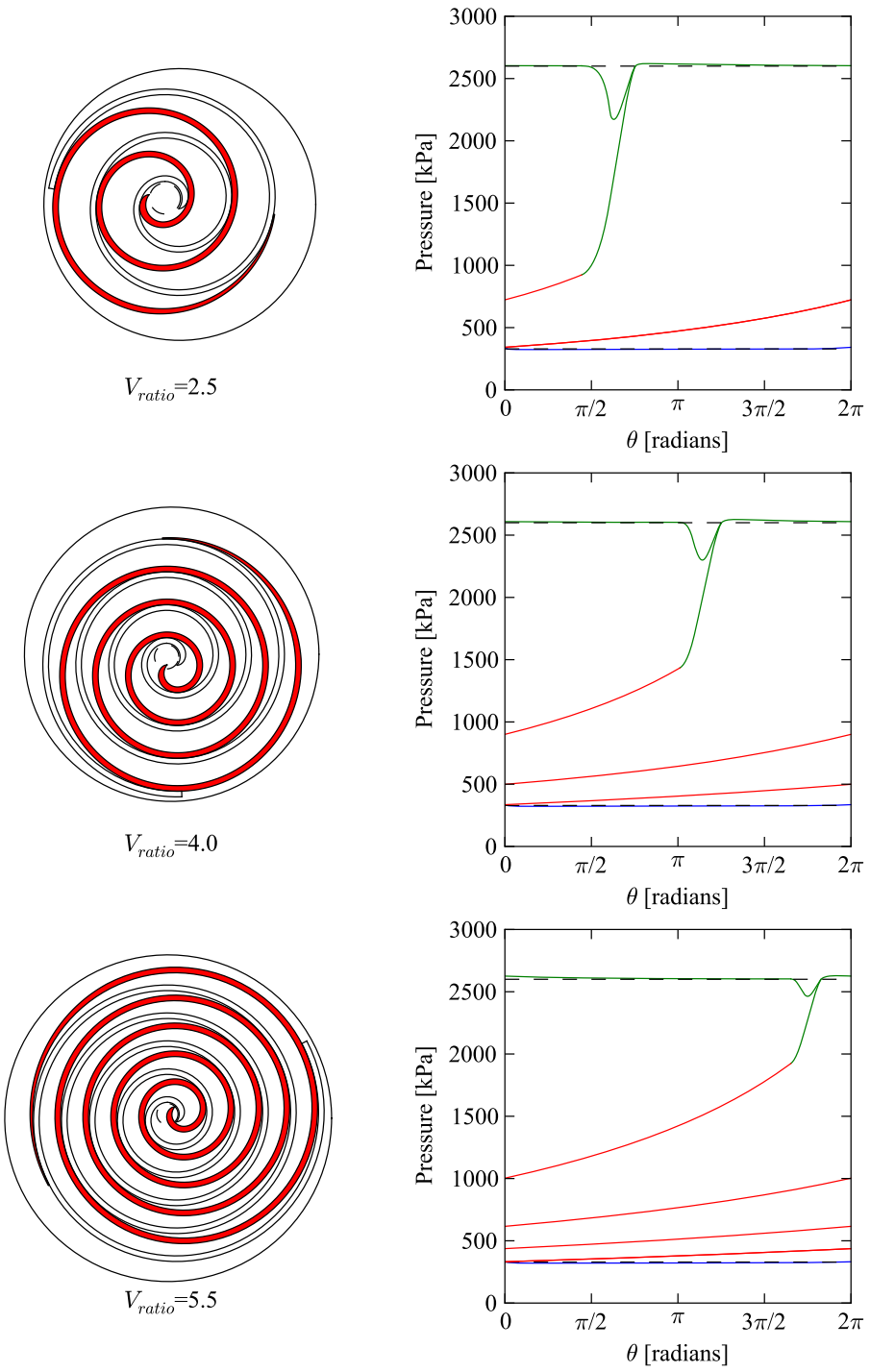
$$\frac{dV_c}{d\theta} = \frac{V_{disp}}{V_{ratio}(-2\phi_{os} - 3\pi - t_s/r_b)} \tag{7.22}$$

which shows that for a given displacement and scroll wrap thickness, as the volume ratio is increased, the derivative of the compression chamber volume goes down, resulting in a more gentle compression process.

This gentle compression at higher volume ratios is one of the strong motivating factors for the use of variable-wall-thickness scroll wraps since they allow for large volume ratios in compact form factors. The geometric analysis is significantly more complex for variable-wall-thickness scroll wraps than for the constant-wall-thickness scroll wrap. The use of variable-wall-thickness scroll wraps might be one means of achieving the large volume ratios needed for the low source temperature R410A heat pump.

In conclusion, as the volume ratio is increased, the isentropic efficiency increases, but so does the difficulty of manufacture. Weighing all these concerns, it seems a

volume ratio of approximately 4 would be a reasonable compromise geometry. A volume ratio of 4 would yield a compressor with a better efficiency than the baseline compressor - even with the addition of 56% oil by mass. And a volume ratio of 4 would not be a significant jump from compressors currently available on the market.



(a) Scroll Wraps (b) Pressure-crank angle plots

Figure 7.14. Scroll wraps and pressure-crank angle plots for range of volume ratios (R410A, POE oil, p_1=329.5 kPa, p_2=2600.3 kPa, T_1=35.3°C, x_l=0.557).

CHAPTER 8. LIQUID-FLOODED EXPERIMENTAL TESTING OF R410A HERMETIC SCROLL COMPRESSOR

Chapter 3 showed that there was a significant potential improvement in system efficiency for heat pumps operating with the refrigerant R410A with oil flooding and regeneration. In addition, Chapter 6 showed that open-drive compressors can accept significant amounts of oil flooding, though they do lose some efficiency. This chapter provides experimental results for the performance of a commercial R410A scroll compressor for residential heat pumping applications with oil injection.

8.1 Experimental System

The scroll compressor was installed in a hot-gas bypass load stand for performance testing with oil injection. Figure 8.1 shows a schematic of a simplified hot gas bypass stand. Refrigerant is compressed from state point 1 to to state point 2 in the compressor, then throttled down to an intermediate pressure at state point 3. At state point 3, some fraction of the flow is throttled through a bypass valve to state point 6. The remaining refrigerant is condensed in the condenser to state point 4, at which point it is throttled from the intermediate pressure down to the suction pressure. This cooled two-phase refrigerant at state point 5 is then mixed with the bypass stream and enters the compressor once again. A pressure-enthalpy plot for the hot-gas bypass stand is shown in Figure 8.2.

The benefit of using the hot-gas-bypass-type load stand is that the intermediate pressure stays relatively constant at a pressure corresponding to a saturation temperature slightly above the condenser water inlet temperature. The exact intermediate pressure depends on the refrigerant charge level and heat transfer in the condenser, but is typically on the order of 5 K above the water inlet temperature.

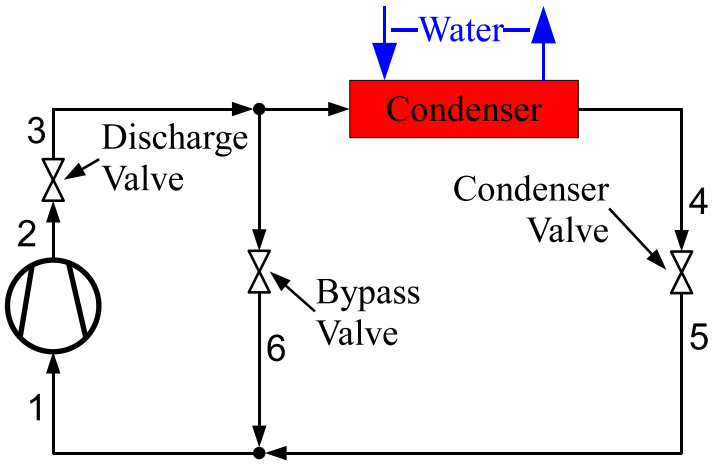


Figure 8.1. Simplified schematic of a standard hot-gas bypass load stand.

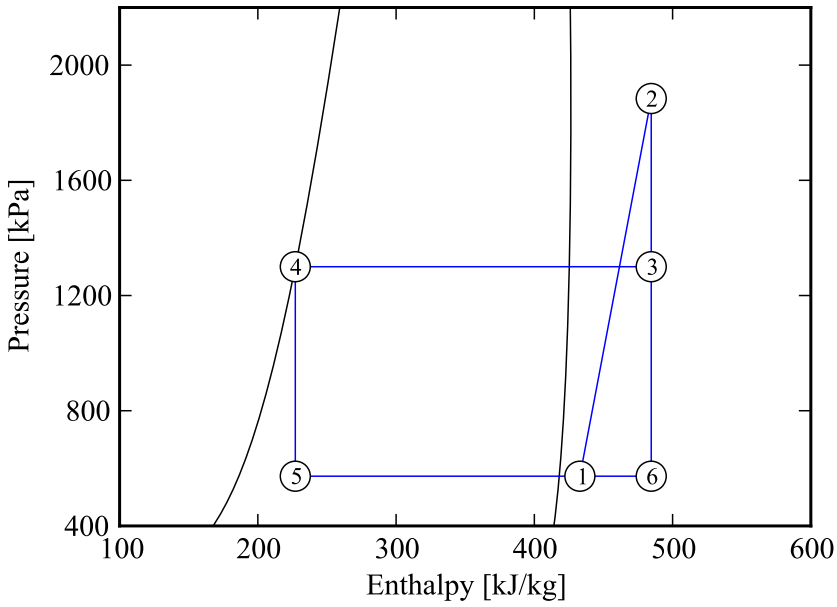


Figure 8.2. Pressure-Enthalpy plot for a conventional hot gas bypass stand.

For the testing conducted here, a few modifications were made to the standard hot-gas-bypass configuration. Two oil loops were added to the stand - a primary oil loop as well as an oil injection loop. Figure 8.3 shows a schematic representation of the modified system. The primary oil separator is used to store oil for return to the

sump of the compressor when the oil in the compressor is depleted. The metering valve labeled MV3 can be opened to add oil to the suction gas stream going into the compressor and refill the compressor oil sump.

The second oil separator is used to process the oil that is injected into the compressor for the purposes of achieving a more-isothermal compression process. When the oil-refrigerant vapor stream exits the compressor, the flow can be balanced between the two separators. In general, only one separator was used at a time. For the oil injection loop separator, a two-phase mixture of oil and refrigerant vapor enter the separator, and essentially pure vapor exits out the top of the separator. Oil with some amount of solved refrigerant then exits the separator from a dip-tube. The oil-refrigerant mixture then is cooled in the oil cooler and throttled through a set of metering valves in order to control the oil injection mass flow rate. The flow rate of injected oil is measured with a Coriolis-type flow meter.

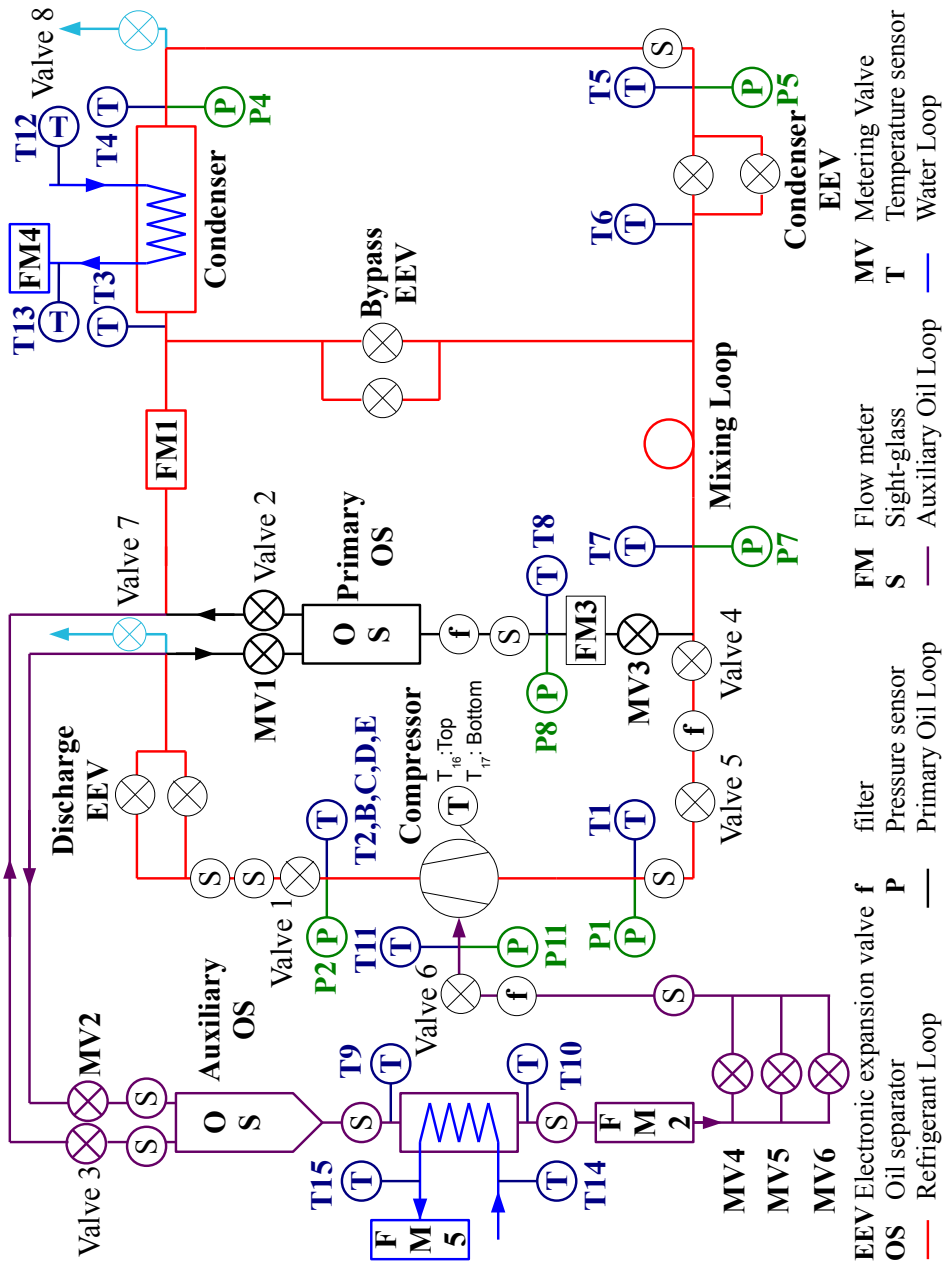


Figure 8.3. Schematic of liquid-flooded scroll compressor stand.

8.2 System Components

Figure 8.4 shows an overview photo of the test rig used to test the R410A compressor. All valves that require user intervention are accessible from the front of the rig, as are the electrical disconnects and switches to control the power to the instrumentation and control loops. The system was constructed such that the compressor can be readily changed without removing the refrigerant from the entire system.

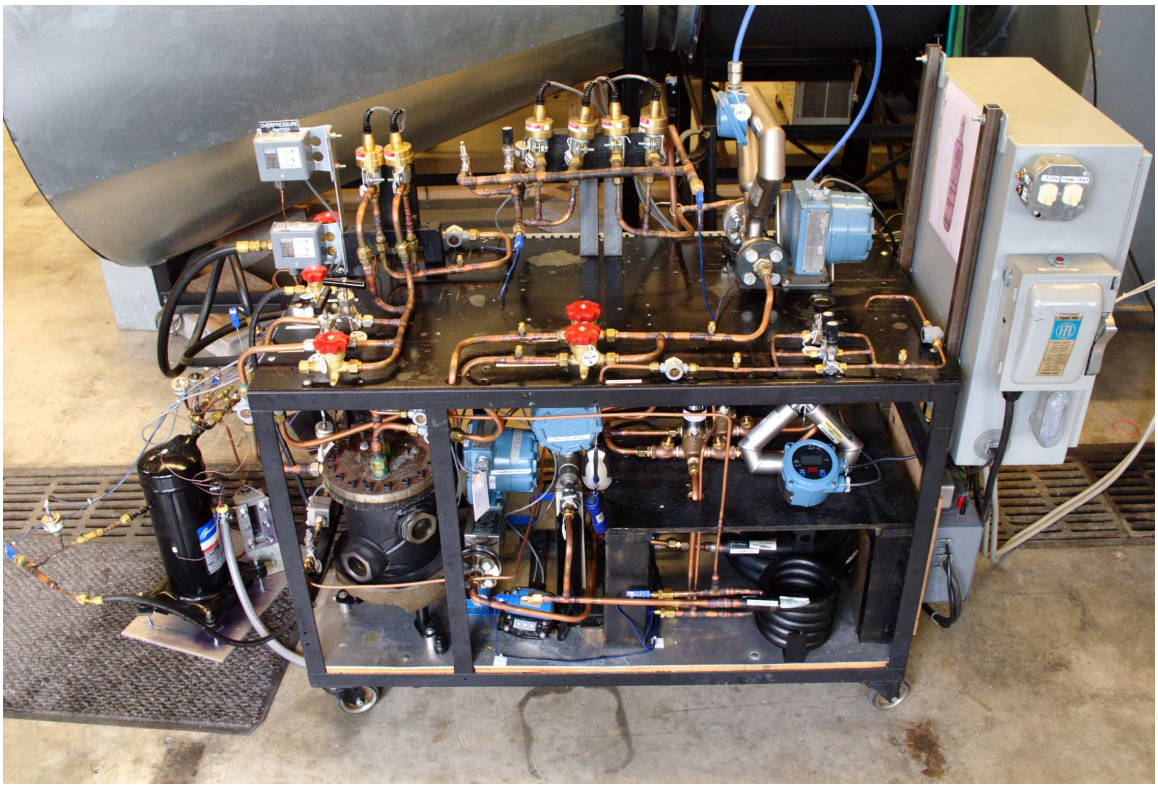


Figure 8.4. Overview of R410A test stand.

8.2.1 Compressor

The compressor utilized for the experimental testing with oil injection is an experimental air-conditioning Copeland compressor (model ZPI29K5E) obtained by adding vapor injection ports to a conventional ZP29K5E compressor. Figure 8.5 shows the

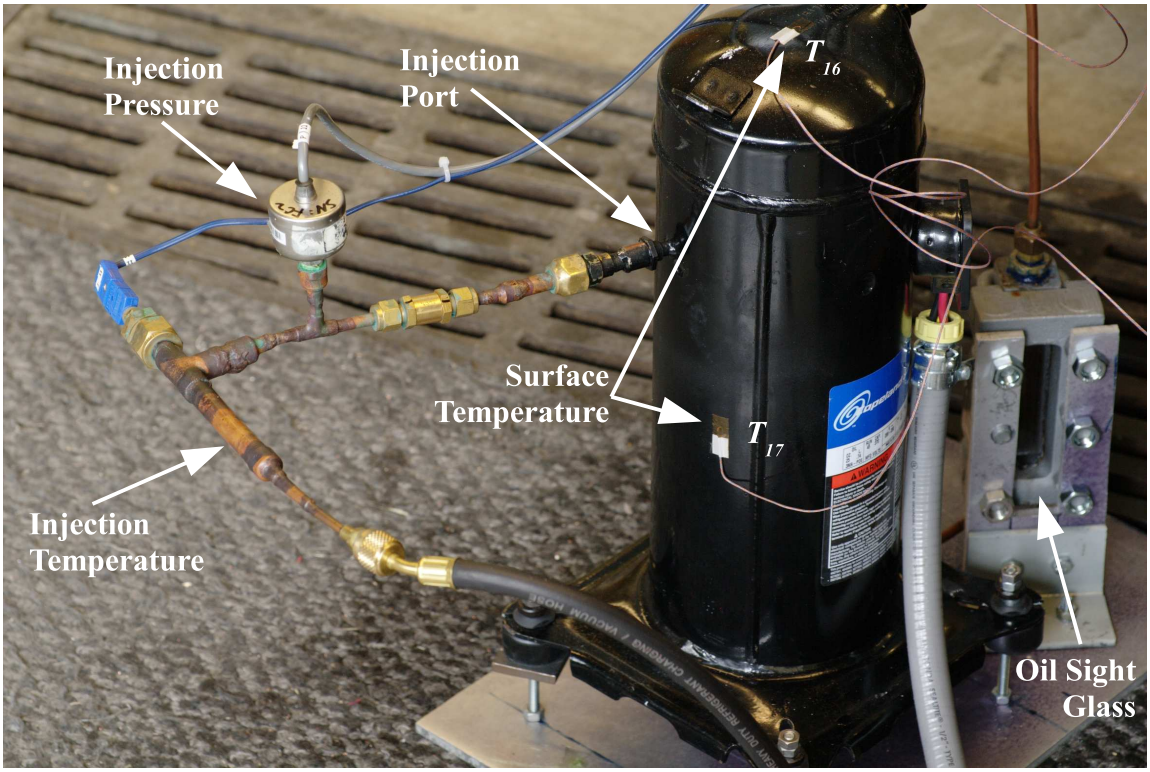


Figure 8.5. Compressor and sight glass in parallel.

compressor installed in the system. The vapor injection ports are located such that they open to the compression chambers just after the compression pocket have been sealed off from the suction chambers. The product information for the ZP29K5E compressor gives a displacement rate of 5.8 m^3/h at 3500 RPM, or a displacement per rotation of 27.6 cm^3.

This compressor is designed for air-conditioning applications, and as a result has a decrease in efficiency at lower evaporating temperatures due to the mal-adjustment of the volume ratio for the imposed pressure ratio, among other design considerations. The compressor runs on a 220 V single-phase supply and rotates at 3500 RPM under load.

The compressor was further modified in order to be able to readily measure the oil level in the sump of the compressor. A small hole was drilled in the bottom of the compressor shell at the dead center location, and this tap was connected to a sight

glass in parallel with the compressor. The top of the sight glass was connected to the suction line to equalize the pressure in the sight glass since the compressor was a low-pressure shell type compressor. Therefore the oil levels in the sight glass and the compressor shell should be equal.

The oil employed in this experimental testing program is a polyol-ester (POE) oil, with the model number Copeland 32-3MAF. The manufacturer recommends this oil for this compressor when operating with refrigerant R410A.

8.2.2 Condenser And Oil Cooler

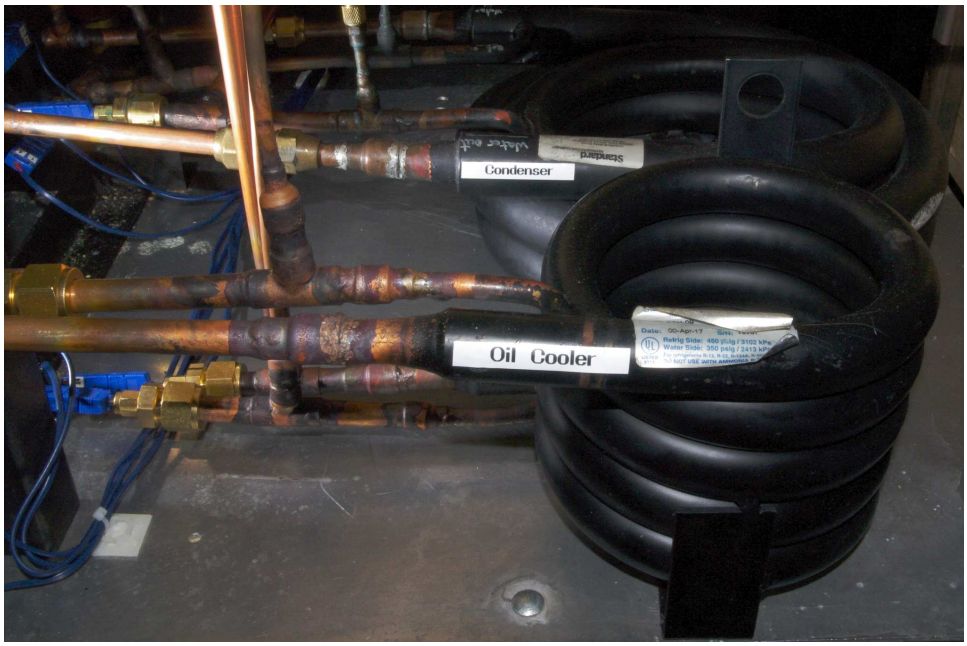


Figure 8.6. Oil cooler and condenser.

The condenser and oil cooler were tube-in-tube type heat exchangers, as seen in Figure 8.6. The oil cooler was a Standard Refrigeration water-cooled condenser with model number SCH-09 with a nominal rating of 0.75 HP. The condenser was a Standard Refrigeration water-cooled-condenser with model number SCS-24 with nominal rating of 2.0 HP. With a 0.126 kg/s (2 gallon per minute) water flow, a 10°C

inlet water temperature, and a 36.6°C condensing temperature, the SCS-24 condenser is capable of a cooling capacity of approximately 7.68 kW based on published capacity data. The condenser only needs to remove the heat added to the system from the compressor, but ambient heat transfer to and from the rig will alter the condenser heat rejection. Without oil injection, roughly 75% of the energy added to the system by the compressor was removed in the condenser.

8.2.3 Oil Separators

The oil separators employed for the testing of the liquid-flooded R410A compressor are the same as were used in the Liquid-Flooded Ericsson Cycle testing. A description of the oil separators can be found in Section 6.1.1. The separators seem to be well-sized for the flow rates experienced in this system. It was not possible to see any bubbles of vapor in the sight glasses installed on the oil outlet lines from the separators. In the separator, it was often seen that there was a fog of atomized oil droplets in the refrigerant phase (as in Figure 8.7), but the vapor outlet at the top of the separator did not appear to suffer from the same problem.

8.2.4 Valves And Valve Controllers

The load stand has a large number of valves installed in order to control the operation of the stand and precisely impose the suction state and the discharge pressure for the compressor. At the discharge of the compressor, two refrigerant electronic expansion valves are installed in parallel to control the compressor discharge pressure. These valves are Sporlan model numbers SEI-6 and SEI-2. These valves are controlled with an Arduino microcontroller that interfaces with a Labview VI that controls the system. For the bypass line, two electronic expansion valves are used in parallel; they are Sporlan models SEI-6 and SEI-2. The bypass valves are also controlled by the Labview VI. The valves on the outlet of the condenser are two Sporlan electronic expansion valves, models SEI-3.5 and SEI-2 as well as two needle valves

Figure 8.7. Oil fog visible in oil separator sight glass.

(Swagelok model SS-6L-MH with maximum C_v of 0.16 and Swagelok model SS-SS4 with maximum C_v of 0.004). It had been hoped initially that the electronic expansion valves installed on the condenser line could have been used to control the superheat, but they were severely oversized, and the needle valves were ultimately used with manual control to control the compressor superheat.

After the flow passes through the discharge valves, there are two Swagelok regulating valves (both model SS-18RS8 with maximum C_v of 1.8) that can be used to balance the two-phase flow between the two separators. Both valves have positive shut-off so that all the flow can be diverted to one of the separators.

On the oil loops, there is a small needle valve on the return line from the primary oil separator (Swagelok model B-4MG2 with maximum C_v of 0.03) and for the oil injection line there are three needle valves in parallel (Swagelok models B-4MG with maximum C_v of 0.03, B-4L-MH with maximum C_v of 0.16, and SS-1RS6 with max-

imum C_v of 0.73). The large number of valves allows for very fine control of the oil return and injection rates.

The water temperature into the oil cooler and the condenser are controlled with the use of mixing valves that are used for residential showers. Each valve is fed with building hot water and building cold water after passing through a particulate filter in order to remove the large amount of scale and rust present in the laboratory water supply. The hot or cold water inlet valves for each shower valve can be manually closed in order to yield full hot or cold water. The condenser was always supplied with only cold water, though the inlet water temperature to the condenser could have been increased in order to increase the intermediate pressure of the rig. The water inlet temperature to the oil cooler was varied from full cold to full hot in order to control the oil injection temperature.

8.2.5 Measurement Devices

The temperatures at all points in the system were measured with Omega T-Type sheathed thermocouples fully inserted in the flow. All the thermocouples were checked against two reference temperatures - the freezing point and boiling point of distilled water at ambient pressure. All of the thermocouples agreed with the reference temperatures to within 0.2°C.

At the discharge of the compressor it was thought initially that there might be significant thermal non-equilibrium effects based on the thermal non-equilibrium challenges experienced in the Liquid-Flooded Ericsson Cycle testing. For that reason, a five-thermocouple system was installed at the discharge of the compressor in the hopes of capturing any thermal non-equilibrium effects if they were present. Two thermocouples were surface mounted on the tube outer wall, and three were installed in the flow, with one pointed upwards and one pointed downwards. Figure 8.8 shows a photograph of the configuration of thermocouples at the discharge of the compressor prior to installation. The surface mounted thermocouples were wrapped with ap-

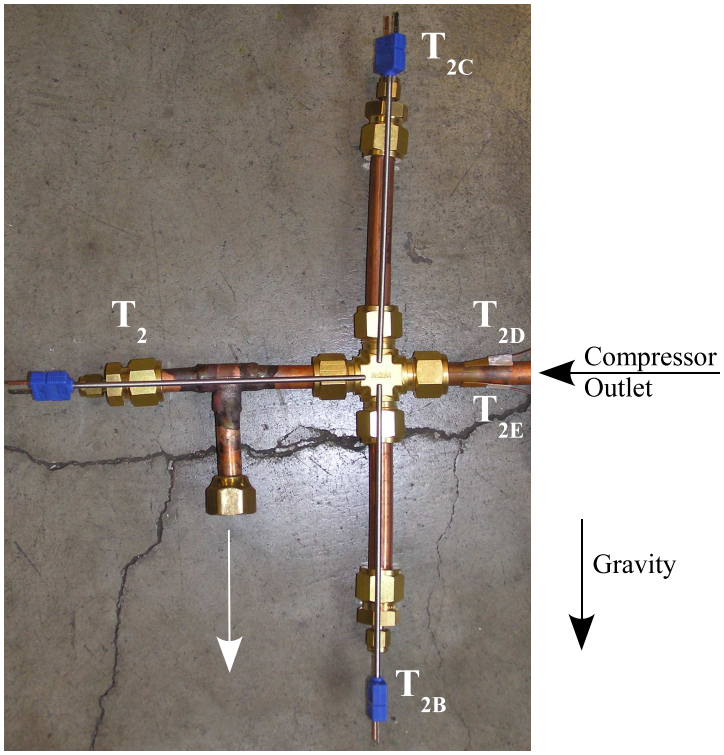


Figure 8.8. Thermocouples installed at the discharge of the compressor.

proximately 1 cm of insulation after installation. Analysis of the experimental data suggests that good thermal equilibrium is achieved at the outlet of the compressor due to the highly turbulent flow in the compression process.

Furthermore, two thermocouples were installed on the shell of the compressor in order to obtain an accurate shell temperature. One thermocouple was installed on the body of the compressor shell (as seen in Figure 8.5), and the other thermocouple was installed on the top of the compressor shell, dead-center.

The pressure transducers were calibrated against a reference pressure transducer with accuracy of 3.44 kPa. After calibration, all the pressure transducers were within tolerance of the reference transducer.

The refrigerant mass flow rate was measured with a MicroMotion Coriolis mass flow meter (model CMF050) with accuracy of 0.35% and full scale range of 1.888 kg/s. The oil injection flow rate was measured with a MicroMotion model CMF025

flow meter with full scale range of 0.6055 kg/s and accuracy of 0.05% for flow rates above 0.030275 kg/s, and an absolute accuracy of 1.513x10^{-5} kg/s below 0.030275 kg/s (5% of full scale). The oil injection flow meter can also measure the density with an accuracy of 0.2 kg/m^3. The condenser water flow was measured with a MicroMotion Coriolis mass flow meter (model R025) with full scale range of 0.755 kg/s and accuracy of 0.5%. The oil cooler water flow rate was measured with a MicroMotion flow meter (model DH025) with full scale range of 0.1888 kg/s and accuracy of 0.15%. The oil return mass flow rate was measured with a GPI gear-type flow meter (model GM001I2C416) with maximum capacity of 13.2 gallons/hour.

The electrical power of the compressor was measured with a Scientific Columbus model XL-525-A2 power meter with accuracy of 0.2% of the reading.

Table 8.1 Summary of measurement devices and uncertainties.

Measurement	Device	Uncertainty
Temperature	T-Type Thermocouple	0.2 K
Pressure	Setra Model 207 & Omega Model PX176	4 kPa
Mass Flow Refrigerant	MicroMotion Model CMF050	0.35%
Mass Flow Oil Injection	MicroMotion Model CMF025	1.5x10^{-5} kg/s
Mass Flow Oil Cooler	MicroMotion Model DH025	0.2%
Mass Flow Condenser	MicroMotion Model R025	0.5%
Electrical Power	Scientific Columbus Model XL-525-A2	0.2%

8.2.6 System Control And Operation

All the electronic expansion valves were wired into motor shields connected to Arduino Duemilanova microcontrollers. The microcontrollers provide the hardware interface to the valves. The Arduino microcontrollers in turn interface via a USB/serial connection with Labview user interface on the data acquisition computer.

Proportional/Integral/Differential (PID) controllers were programmed into the Arduino controllers, but it was found ultimately to be significantly easier to do the PID control within Labview and pass valve opening values to the controllers through the serial connection. The PID controller in Labview is structured so that the valves can be operated in manual mode at startup, or whenever necessary. The basic control scheme for the rig is as shown in Figure 8.9. The suction and discharge pressures for the compressor are measured, and from the suction and discharge pressures, the suction and discharge dew temperatures are calculated corresponding to the suction and discharge pressures respectively. The error between the setpoint for the dew temperature and the measured dew temperature is the input value to the PID controller. Ultimately the goal of the PID controller is to drive the measured dew temperature and the setpoint dew temperature together by opening or closing the valves.

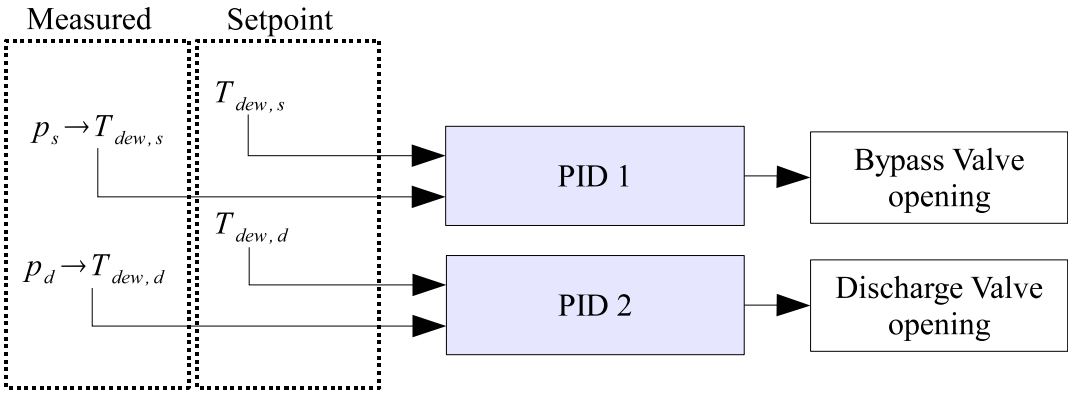


Figure 8.9. PID controller flowchart.

The most direct effect of closing the discharge valves is to increase the discharge pressure, although the valve position of the discharge valves also has an impact on superheat and compressor suction pressure. The compressor suction pressure is largely driven by the valve opening of the bypass valves, and the compressor superheat is governed by the condenser valve opening.

The Labview VI was linked with REFPROP (Lemmon, 2010) to calculate the refrigerant properties. R410A is a 50%/50% (by mass) blend of R32 and R125, and

the full mixture properties were employed in the calls to REFPROP. This caused REFPROP to not converge in its calculation for dew temperatures at some points.

In order to arrive at a given state point, the following operating procedure is employed:

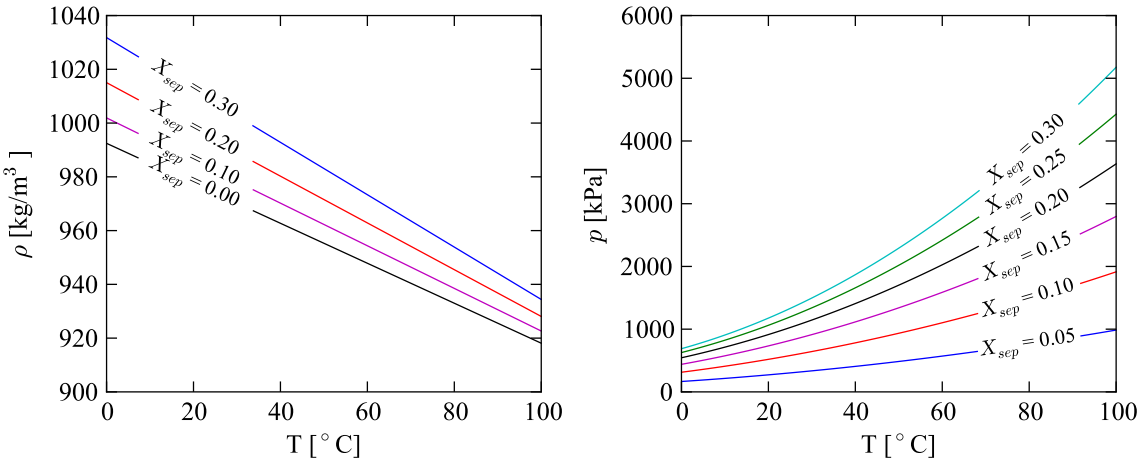
- Turn on water to rig and wait for the temperature to stabilize
- Fully open all electronic expansion valves for discharge and bypass lines and set to manual control
- Fully close all condensing loop valves
- Check that all hand valves that should be open are open, and all hand valves that should be closed are closed
- Turn on compressor
- Use manual control of the discharge and bypass valves to get dew temperatures to within 2K of desired value, then turn on PID mode
- At the same time, very slowly open the needle valves on the condensing loop in order to decrease the superheat
- If oil injection is being used, open the oil injection needle valves and turn on the oil cooler if desired
- Wait for steady-state operation to be reached - some fine adjustment may be necessary to reach the exact state point desired

8.3 Data Reduction

8.3.1 Solubility

In the oil separators, oil and the refrigerant come to a thermodynamic equilibrium, the result of which is that some amount of refrigerant remains dissolved in the oil. In general, the solubility mass fraction is a function of both temperature and pressure. Figure 8.10 shows the results for the equilibrium mixture density and pressure for a mixture of R410A and 3MAF POE oil. For a given temperature, as the pressure increases, the amount of refrigerant dissolved in the oil increases. For a given pressure,

as the temperature increases, the amount of refrigerant dissolved in the oil decreases. Using the measured pressure and temperature in the oil injection oil separator it is therefore possible to estimate the mass fraction of refrigerant dissolved in the oil (X_{sep}). The pressure of the oil injection oil separator is assumed to be equal to the intermediate pressure measured at the inlet of the condenser. The net result is that a R410A mass fraction of approximately 0.15 is dissolved in the oil at the outlet of the separator and subsequently re-injected into the compressor through the oil injection port.



(a) Density

(b) Pressure

Figure 8.10. Thermodynamic properties of R410A-3MAF oil mixture as a function of temperature and equilibrium refrigerant mass fraction from manufacturer data (X_{sep} is mass fraction of refrigerant).

8.3.2 Calculations

The dew temperatures for the compressor are found from the measured suction and discharge pressures. Then the compressor inlet superheat is defined as

$$\Delta T_{sh} = T_1 - T_{dew,s} \tag{8.1}$$

The mass flow rates of refrigerant ($\dot{m}_{\text{ref}}$) and oil with dissolved refrigerant ($\dot{m}_l$) are directly measured and thus the liquid mass fraction can be given by

$$x_l = \frac{\dot{m}_{\text{ref}}}{\dot{m}_{\text{ref}} + \dot{m}_l} \tag{8.2}$$

The total isentropic power is given by the sum of the isentropic power required to compress the refrigerant to the discharge pressure and the power needed to isentropically compress the injected mixture of oil and refrigerant oil to the discharge pressure or

$$\dot{W}_i = \dot{m}_{\text{ref}}(h_{2s} - h_1) + \dot{m}_l(h_{2s,inj} - h_{11}) \tag{8.3}$$

where h_{2s} and $h_{2s,inj}$ are the isentropic compression enthalpies from the suction and injection pressures respectively to the discharge pressure. The injected mixture is modeled based on homogeneous oil-refrigerant mixture properties. Therefore the overall isentropic efficiency can be given by

$$\eta_{oi} = \frac{\dot{W}_i}{\dot{W}_{\text{el}}} \tag{8.4}$$

As in the analysis for the Liquid-Flooded Ericsson Cycle testing presented above, the uncertainties of all the derived parameters were calculated based on an automatic uncertainty calculation routine. All the data reduction calculations were embedded in an outer loop that calculated the uncertainties of each parameter by numerical differentiation. Thus the calculations can be freely modified or other output parameters can be calculated, and the experimental uncertainties will be automatically updated.

8.3.3 Test Matrix

In order to develop an understanding of the performance of the compressor with oil flooding, the compressor was run at a selection of test points. Since the motivation for this set of experimental testing is to apply liquid-flooding and regeneration to low-temperature heat pumping applications, low evaporation temperatures were employed. The goal was to test the compressor down to suction dew temperatures

as low as -20°C. Unfortunately flow instabilities in the flow through the condenser at suction dew temperatures lower than -10°C made it impossible to achieve a stable superheat at a suction dew temperature of -20°C. Altering the valve C_v by less than 0.0001 resulted in a change in superheat of 0.3°C. It would seem that at a suction dew temperature of -20°C, the two-phase flow in the condenser is on the cusp of transition between flow regimes, as it is possible to hear different flow patterns with the use of a screwdriver applied to the tube at the outlet of the condenser valves. At higher evaporation temperatures, oil management becomes difficult because the higher flow rates of refrigerant through the compressor result in larger amounts of oil entrainment in the refrigerant passing through the compressor shell. As a result, the compressor sump needs to be refilled so frequently that it is impossible to achieve steady-state operation at higher suction dew temperatures. As a result, the only suction dew temperature that can be used in this configuration is -10°C.

Table 8.2 R410A Test Matrix.

Config.	$T_{dew,s}$ °C	$T_{dew,d}$ °C	ΔT_{sh} °C	Oil
A	-10±0.1	30±0.1	11.1±0.1	Cooled to 25±3°C
B	-10±0.1	43.3±0.1	11.1±0.1	Not actively cooled
C	-10±0.1	43.3±0.1	11.1±0.1	Cooled to 35°C
D	-10±0.1	43.3±0.1	11.1±0.1	Cooled to 15°C
E	-10±0.1	43.3±0.1	15±0.1, 25±0.1	Cooled to 35°C, x_l=0.3

The configurations shown in Table 8.2 were used to test the compressor. For configurations A through D, the oil flow rate was varied in order to yield nominal increments in the oil mass fraction of 0.05. The highest mass fraction in each set was the maximum amount of oil that could be injected in the compressor with all the oil injection valves fully open. At the highest oil mass flow rates, there was up to a 2 bar pressure difference between the system intermediate pressure and the injection port

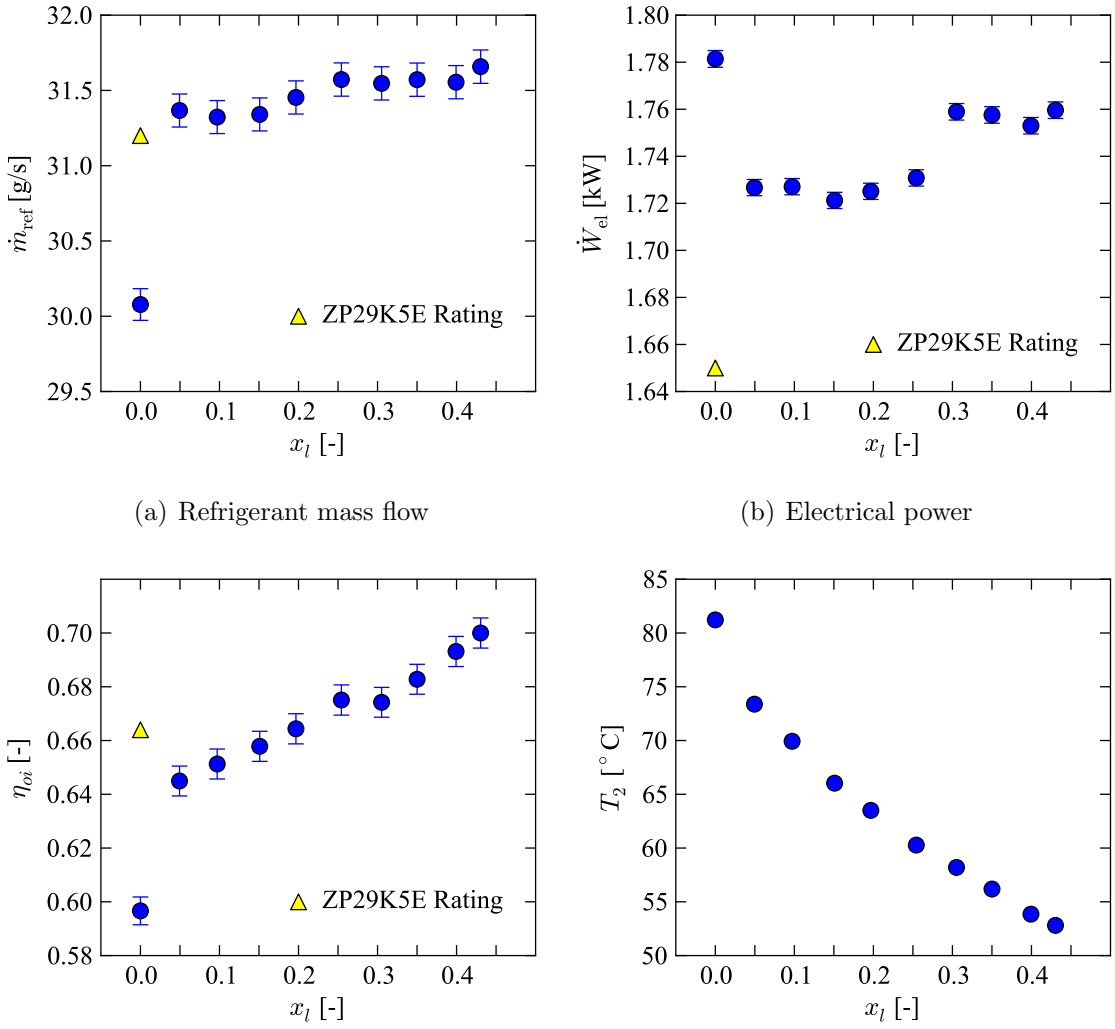
pressure. The superheat was set at 11.1 K for consistency with the rating data for the ZP29K5E compressor. In practice, in the liquid-flooded application, the superheat will be much higher, and the inlet temperature to the compressor will be near the heat sink temperature due to regeneration. The tested compressor could not operate at the very high superheat states because it was impossible to inject enough oil to keep the discharge temperature sufficiently low. Configuration E was meant to try to understand the performance of the compressor at higher superheat.

At each state point, at least 5 minutes of steady-state data was acquired. The condensing and evaporating pressures were very stable because they were being dynamically controlled with the PID controllers, while the superheat tended to vary because of slight variations in the inlet water temperature to the condenser. The data was then post-processed with a Python script to calculate all parameters not directly calculated and the uncertainties of all parameters. The pertinent parts of the Python code can be found in Appendix E.2.

8.4 Experimental Results

Figure 8.11 shows the performance of the the compressor at the -10°C suction dew temperature / 30°C discharge dew temperature configuration (configuration A). The manufacturer data for the ZP29K5E compressor at this state point without injection ports yields an isentropic efficiency of 66.4%, electrical power of 1650 W and refrigerant mass flow rate of 31.2 g/s.

When no oil injection is used (x_l=0), the compressor performance is worse than the manufacturer rating data for the ZP29K5E compressor. This decrease in performance is due to the re-expansion losses of the gas in the injection lines. During each rotation of the compressor the gas in the oil injection lines is compressed, and then subsequently re-expands, causing irreversibilities. The irreversibilities are manifest as a heating source term which result in a higher discharge temperature, and less mass flow rate due to suction gas heating. In addition, due to a few adverse lubri-



(a) Refrigerant mass flow

(b) Electrical power

(c) Overall isentropic efficiency

(d) Temperature ratio

Figure 8.11. Performance of oil-injected compressor for $T_{dew,s}$=-10°C and $T_{dew,d}$=30°C with and without oil injection (Error bars: uncertainty).

cation events when the oil sump ran out of oil during the oil refilling process, some mechanical damage would seem to have been done to the bearing system.

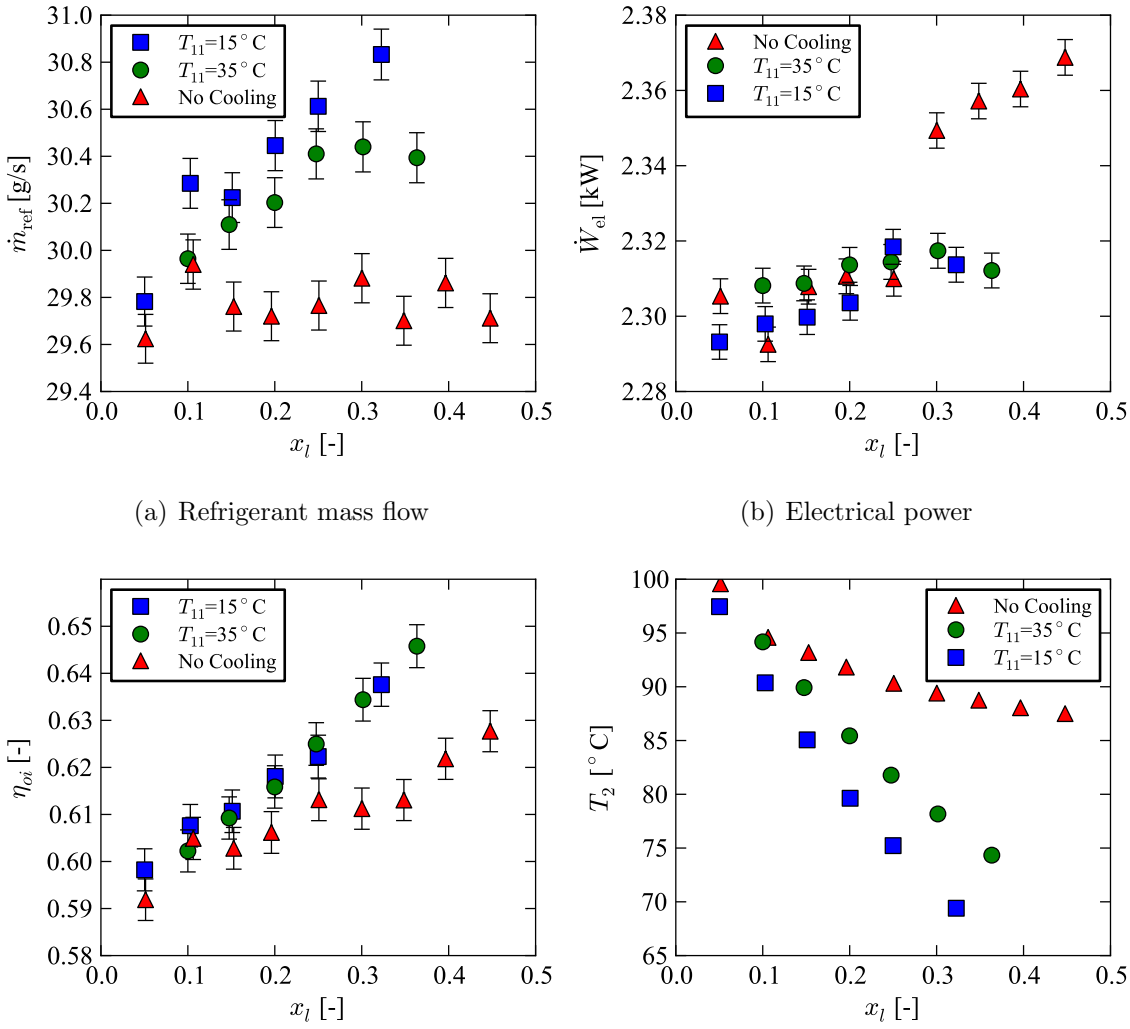
Once even a small amount of oil injection is introduced, the mass flow rate jumps back to the ZP29K5E rating mass flow rate. This is due to the fact that as soon as there is some flow of oil in the injection lines, they are then full of incompressible

oil rather than compressible refrigerant vapor, and the re-expansion losses from the injection lines are eliminated. The power does not fully recover to the ZP29K5E data, largely due to the mechanical damage caused by the adverse lubrication events. As the oil injection rate is increased, the refrigerant mass flow rate increases slightly. This increase in refrigerant mass flow rate occurs due to a decrease in heat transfer to the gas in the suction pocket because the rest of the compression process is more isothermal.

As increasing amounts of oil are injected, the refrigerant mass flow rates continue to increase. The discharge temperature of the compressor decreases monotonically with larger oil mass fractions due to the refrigerant's heat of compression being transferred to the oil. The mechanical power remains effectively constant until an oil mass fraction of 0.30. Between an oil mass fraction of 0.25 and 0.30 there is a step change in the electrical power. This same effect is seen in the other tests described below. The physical mechanism behind this increase in power is not clear, but it has been surmised that it might be due to a dis-engagement of the radial compliance mechanism due to large oil film forces, or a transition in flow regime resulting in a large change in discharge pressure drops. Or the increase in power could be a result of a disturbance in the lubrication, even though there was a sufficient amount of oil in the compressor sump. The introduction of two-phase flow effects makes analysis of the compression process and all the physical phenomena that contribute to the electrical power more difficult.

As oil is injected into the compression process, the overall isentropic efficiency of the compressor generally increases monotonically with increasing amounts of oil injection. The only exception is between oil injection mass fractions of 0.25 and 0.3, where there is a small step in the overall isentropic efficiency due to the discontinuity in the electrical power.

At a discharge dew temperature of 43.3°C (configurations B through D), the performance of the compressor with oil injection is qualitatively similar to the performance at a discharge dew temperature of 30°C. Figure 8.12 shows the results for the



(a) Refrigerant mass flow

(b) Electrical power

(c) Overall isentropic efficiency

(d) Temperature ratio

Figure 8.12. Performance of oil-injected compressor for $T_{dew,s}$=-10°C and $T_{dew,d}$=43.3°C with and without oil injection (Error bars: uncertainty).

compressor with varying amounts of oil injection, as well as varying the amount of cooling of the oil prior to injection. The nominal rating for the ZP29K5E compressor at this state point is an electrical power of 2290 W, overall isentropic efficiency of 0.596, and refrigerant mass flow rate of 29.6 g/s.

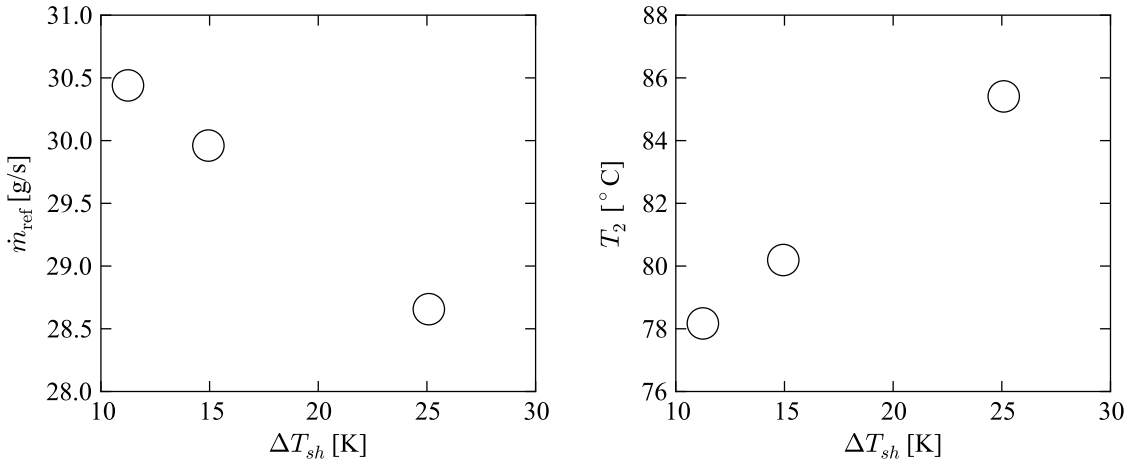
When the oil is not actively cooled prior to injecting it into the compressor, the oil is not as able to provide cooling of the refrigerant during the working process - in fact the injected oil is much hotter than the fluid in the compression pocket when it begins being in contact with the injected oil. As a result, the discharge temperature of the compressor sees only a small decrease from the oil injection. The refrigerant mass flow rate is effectively constant. Again, the sharp step change in electrical power between oil mass fractions of 0.25 and 0.3 is experienced.

When the oil is cooled to 35°C prior to injection into the compressor, the refrigerant mass flow rate increases due to the cooler inlet temperature of the oil. The discharge temperature of the compressor sees a large decrease in temperature from 111.2°C without oil injection to 74.3°C at an oil mass fraction of 0.363. This testing condition simulates cooling the oil against the sink temperature for an air-to-air heat pump application where the sink temperature is the interior of the building to be heated.

Further cooling the oil to an injection temperature of 15°C, the benefits to compressor performance are increased. There is a slight improvement in the refrigerant mass flow rate compared to the 35°C oil injection temperature, but the primary influence is on the discharge temperature of the compressor, which is decreased down to 69.4°C. This condition was designed to simulate cooling the oil against the outdoor temperature, though colder temperatures could not be achieved since the oil cannot be cooled below the cold water inlet temperature.

At both oil injection temperatures of 15°C and 35°C, the overall isentropic efficiency increases monotonically with an increase in oil injection mass fraction. When the oil is not cooled, the overall isentropic efficiency also increases monotonically with the oil injection mass fraction, with the exception of the same discontinuity between oil injection mass fractions of 0.25 and 0.30. The discontinuity in the overall isentropic efficiency without oil cooling is due to the discontinuity in the electrical power.

Figure 8.13 shows the performance of the compressor with an oil injection mass ratio of 0.30 for varied suction superheat. As the superheat is increased, the refrig-



(a) Refrigerant mass flow rate

(b) Discharge temperature

Figure 8.13. Performance of compressor for varied superheat ($T_{dew,s}$=-10°C, $T_{dew,d}$=43.3°C, x_l=0.30, T_{11}=35°C).

erant mass flow rate decreases due to the decrease in refrigerant density at higher superheat. Thus when designing a compressor for practical application to oil flooding with regeneration systems, larger compressor displacements would be needed for the same system heating or cooling capacity due to the extremely high superheat. The discharge temperature of the compressor also increases due to the higher suction temperature, but the increase in discharge temperature is less than the increase in suction superheat. The overall isentropic efficiency is constant to within the experimental uncertainty.

8.5 Summary

An air-conditioning compressor for refrigerant R410A with a vapor-injection port has been tested with oil injection. The compressor was installed in a well-instrumented testing facility in order to measure performance with oil injection. It was found that in general the injection of oil results in an increase in refrigerant mass flow rate and overall isentropic efficiency and a decrease in the compressor discharge temperature.

This dataset suggests that designing an efficient compressor with oil injection for application to a low-source-temperature air-source heat pump applications should be possible.

CHAPTER 9. SUMMARY AND RECOMMENDATIONS

9.1 Summary

In this document, a number of models have been developed. The first model presented is for the performance of liquid-flooded air conditioning and heat pump systems. Based on a simple thermodynamic model, the results show that there is a significant benefit to system efficiency possible for a wide range of working fluids. Perhaps most interestingly, both propane and ammonia see large increases in cycle efficiency, even though they already have good cycle efficiency without liquid flooding and are both natural refrigerants. The benefits in cycle efficiency for carbon dioxide are also large, though even after adding liquid flooding and regeneration, the COP of the carbon dioxide system with liquid flooding is still relatively low as compared with other working fluids.

A model is also presented for the geometry of the scroll compressor. This model encompasses analytic solutions for the volumes of the working chambers, the force components from the gas pressure, and the calculation of leakage and primary flow areas, including the discharge port blockage effects. This model is general enough to handle multiple pairs of compression chambers, and provides analytic solutions for some elements of the geometry for which no analytic solutions are available in open literature.

With the geometry of the scroll compressor defined, it is then possible to implement the geometry of the scroll compressor into an overall model for the compressor. This scroll compressor model can handle liquid flooding, but can also run without liquid flooding. An adaptive-step-size solver is used to integrate a set of differential equations in order to determine the power, discharge temperature and mass flow rate, among other output parameters.

The scroll compressor model has been validated and tuned against a set of experimental testing carried out on the Liquid-Flooded Ericsson Cycle. From this data it was possible to show that the compressor model accurately captures the physics of the liquid-flooded compression process. In addition, a few parameters were tuned that are difficult to estimate *a priori*.

The scroll compressor model was then used to optimize nitrogen, R410A, and carbon dioxide scroll compressors for liquid flooded applications. Simple relations for ideal scroll compressor geometry can be obtained. It is predicted that good compressor efficiency can be achieved with liquid flooding.

A hermetic R410A air-conditioning compressor was tested with oil injection up to oil mass fractions of approximately 40%, for which is was seen that the refrigerant mass flow rate increases monotonically with the oil injection rate, and the efficiency of the compressor also increases.

9.2 Recommendations

Based on the results shown for open-drive and hermetic compressors with varied levels of oil injection, it seems clear that scroll compressors can handle oil injection or flooding without any seriously negative effects. In fact, it seems the efficiency of at least the hermetic R410A compressor increases with oil injection due to the decrease of heat transfer irreversibilities. A full liquid-flooded system with regeneration should be constructed to better understand the system-level impacts of the addition of the oil and regeneration. It is expected that the performance of the oil-flooded system should be improved compared to a baseline system without oil flooding.

For an accurate model of liquid-flooded system performance, it is necessary that the understanding of the mixture thermodynamics be improved. The oil-refrigerant mixture modeling employed here neglected much of the complexity of mixtures, and further study could be used to improve the treatment of the mixture's thermodynamic and transport properties.

From a compressor modeling standpoint, some effort could be expended on improving the mechanical losses model. It is quite challenging to accurately model the mechanical losses in scroll compressors due to the range of phenomena and components involved (dry friction, elasto-hydrodynamic lubrication, journal bearings, roller bearings). Even state-of-the-art models struggle to accurately capture the mechanical losses. This is particularly true since it is very difficult to decouple the mechanical losses from the rest of the working process, and as a result it is difficult to validate or improve the mechanical losses model.

Since larger volume ratios are needed for flooded compression than dry compression, it would be useful to add variable-wall-thickness scroll wraps to the geometric modeling. The addition of variable-wall-thickness scroll wraps would add significant challenges in calculations of all geometric parameters and would be a major undertaking.

LIST OF REFERENCES

LIST OF REFERENCES

T. Afjei, P. Suter, and D. Favrat. Experimental analysis of an inverter-driven scroll compressor with liquid injection. In *1992 Compressor Engineering Conference at Purdue University*, 1992.

Air-Conditioning, Heating and Refrigeration Institute. Central air conditioners and air-source heat pumps shipment statistics. URL `http://www.ahrinet.org/ARI/util/showdoc.aspx?doc=630`.

R. Battino, T. R. Rettich, and T. Tominaga. The Solubility of Nitrogen and Air in Liquids. *J. Phys. Chem. Ref. Data*, 13:563–600, 1984.

T. W. Bein and J. F. Hamilton. Computer modeling of an oil flooded single screw air compressor. In *1982 International Compressor Engineering Conference at Purdue University*, pages 127–134, 1982.

I. Bell, V. Lemort, E. Groll, J. Braun, and G. King. Liquid-Flooded Compression and Expansion in Scroll Machines - Part I: Model Development. *Int. J. Refrig.*, Submitted for Publication, 2011.

S. S. Bertsch. Theoretical and experimental investigation of a two stage heat pump cycle for nordic climates. Master's thesis, Purdue University, 2005.

J. Blaise and T. Dutto. Influence of oil injection and pressure ratio on single screw performances at high temperatures. In *1988 International Compressor Engineering Conference at Purdue University*, pages 338–345, 1988.

B. Blunier, G. Cirrincione, and A. Mairaoui. Novel Geometrical Model of Scroll Compressors for the Analytical description of the chamber volumes. In *2006 International Compressor Engineering Conference at Purdue University*, number C074, 2006.

B. Blunier, G. Cirrincione, Y. Hervé, and A. Miraoui. A new analytical and dynamical model of a scroll compressor with experimental validation. *International Journal of Refrigeration*, 32:874–891, 2009.

E. R. Booser, editor. *Tribology Data Handbook*. CRC Press, 1997.

C. Burton and A. Jacobi. Vapor-Liquid Equilibria for R-32 and R-410A Mixed With a Polyol Ester: Non-Ideality and Local Composition Modeling. Technical Report ACRC TR-117, University of Illinois, 1997.

C. Burton, A. Jacobi, and S. Mehendale. Vapor-liquid equilibrium for R-32 and R-410A mixed with a polyol ester: non-ideality and local composition modeling. *International Journal of Refrigeration*, 22:458–471, 1999.

J. W. Bush and W. P. Beagle. Derivation of a General Relation Governing the Conjugacy of Scroll Profiles. In *1992 International Compressor Engineering Conference at Purdue University*, pages 1079–1087, 1992.

J. W. Bush, W. P. Beagle, and M. E. Housman. Maximizing Scroll Compressor Displacement Using Generalized Wrap Geometry. In *1994 International Compressor Engineering Conference at Purdue University*, pages 205–210, 1994.

R. H. Byrd, P. Lu, and J. Nocedal. A Limited Memory Algorithm for Bound Constrained Optimization. *SIAM Journal on Scientific and Statistical Computing*, 16: 1190–1208, 1995.

Y. Chen. *Mathematical Modeling of Scroll Compressors*. PhD thesis, Purdue University, 2000.

Y. Chen, N. Halm, E. Groll, and J. Braun. Mathematical Modeling of Scroll Compressor. Part I- Compression Process Modeling. *International Journal of Refrigeration*, 25:731–750, 2002.

D. Chisholm. *Two-Phase flow in pipelines and heat exchangers*. George Goodwin, London, 1983.

N.-K. Cho, Y. Youn, B.-C. Lee, and M.-K. Min. The Characteristics of Tangential Leakage in Scroll Compressors for Air-conditioners. In *15th International Compressor Engineering Conference at Purdue University*, number 807-814, 2000.

M. R. Conde. Estimation of Thermophysical Properties of Lubricating Oils and their solutions with Refrigerants: An Appraisal of Existing Methods. *Applied*, 16: 51–61, 1996.

L. Creux. Rotary Engine - US Patent No. 801182. 1905.

K. Dinh and W. Lear. Effects of heat transfer from a scroll compressor using heat pipes. *American Society of Mechanical Engineers, Advanced Energy Systems Division (Publication) AES*, 45:71 – 78, 2005. ISSN 10716947.

Z. Duan and R. Sun. An improved model calculating CO_2 solubility in pure water and aqueous NaCl solutions from 273 to 533 K and from 0 to 2000 bar. *Chemical Geology*, 193:257–271, 2003.

J. Fahl. *Entwicklung und Erprobung von Schmierlen für Kälte- und Klimasysteme mit CO_2 als Arbeitsstoff [in German]*. PhD thesis, Ruhr-Universität Bochum, 2002.

Z. Fan and Z. Chen. A calculating method for gas leakage in compressor. In *1994 Compresor Conference at Purdue University*, 1994.

O. Fandiño, E. R. López, L. Lugo, M. Teodorescu, A. M. Mainar, and J. Fernández. Solubility of Carbon Dioxide in Two Pentaerythritol Ester Oils between (283 and 333) K. *J. Chem. Eng. Data*, 53:1854–1861, 2008.

A. Fenghour, W. Wakeham, V. Vesovic, J. Watson, J. Millat, and E. Vogel. The Viscosity of Ammonia. *J. Phys. Chem. Ref. Data*, 24:1649–1667, 1995.

A. Fenghour, W. Wakeham, and V. Vesovic. The Viscosity of Carbon Dioxide. *J. Phys. Chem. Ref. Data*, 27:31–44, 1998.

M. Fujiwara and Y. Osada. Performance analysis of an oil-injected screw compressor and its application. *Int. J. Refrig.*, 18:220–227, 1995.

A. P. Garbarino, editor. *2002 ASHRAE Refrigeration Handbook*, chapter Refrigeration Load, pages 12.1–12.7. American Society of Heating, Refrigeration, and Air Conditioning Engineers, 2002.

J. García, X. Paredes, and J. Fernández. Phase and volumetric behavior of binary systems containing carbon dioxide and lubricants for transcritical refrigeration cycles. *The Journal of Supercritical Fluids*, 45(3):261 – 271, 2008. ISSN 0896-8446. doi: DOI:10.1016/j.supflu.2008.01.022.

V. Geller. Viscosity of Mixed Refrigerants R404A, R407C, R410A, and R507A. In *2000 Refrigeration and Air Conditioning Conference at Purdue*, 2000.

V. Geller, B. Nemzer, and U. Cheremnykh. Thermal Conductivity of the Refrigerant mixtures R404A, R407C, R410A, and R507A. *Int. J. Thermophysics*, 22:1034–1043, 2001.

J. E. Gere. *Mechanics of Materials 5th Edition*. Brooks/Cole, 2001.

H. Getu and P. Bansal. Thermodynamic analysis of an R744-R717 cascade refrigeration system. *International Journal of Refrigeration*, 31(1):45 – 54, 2008. ISSN 0140-7007. doi: 10.1016/j.ijrefrig.2007.06.014.

J. Gravesen and C. Henriksen. The Geometry of the Scroll Compressor. *SIAM Review*, 43(1):113–126, 2001. ISSN 00361445.

N. Halm. Mathematical Modeling of Scroll Compressors. Master's thesis, Purdue University, 1997.

A. Hauk. *Thermo- und Fluiddynamik von synthetischen Schmierstoffen mit Kohlendioxid als Kltemittel in PKW-Klimaanlagen [in German]*. PhD thesis, Ruhr-Universitt Bochum, 2001.

A. Hauk and E. Weidner. Thermodynamic and Fluid-Dynamic Properties of Carbon Dioxide with Different Lubricants in Cooling Circuits for Automobile Application. *Ind. Eng. Chem. Res.*, 39:4646–4651, 2000.

A. Hiwata, N. Iida, Y. Futagami, K. Sawai, and N. Ishii. Performance investigation with oil-injection to compression chambers on CO_2-scroll compressor. In *2002 Purdue Compressor Conference*, number C18-4, 2002.

Y. Huang. Leakage Calculation through clearances. In *1994 Compressor Conference at Purdue University*, 1994.

J. Hugenroth. *Liquid Flooded Ericsson Cycle Cooler*. PhD thesis, Purdue University, 2006.

J. Hugenroth, J. Braun, E. Groll, and G. King. Oil Flooded Compression in Vapor Compression Heat Pump Systems. In *IHR-IRHACE Conference*, pages 492–499, 2006.

J. Hugenroth, J. Braun, E. Groll, and G. King. Thermodynamic analysis of a liquid-flooded Ericsson cycle cooler. *Int. J. Refrig.*, 207:331–338, 2007.

J. Hugenroth, J. Braun, E. Groll, and G. King. Experimental investigation of a liquid-flooded Ericsson cycle cooler. *Int. J. Refrig.*, 31:1241–1252, 2008.

A. Hunold. *VDI Warm Atlas*, chapter D4. 3 Oil-based and Synthetic Heat Transfer Media, pages 419–511. Springer Verlag, 2010. ISBN 3540778764.

J. D. Hunter. Matplotlib: A 2D graphics environment. *Computing In Science & Engineering*, 9(3):90–95, May-Jun 2007.

K. Ignatiev and J.-L. Caillat. Injection System and Method for Refrigeration System Compressor - US Patent US 2008/0078192. 2008.

F. P. Incropera and D. P. Dewitt. *Fundamentals of Heat and Mass Transfer (5th Edition)*. Wiley, 2002.

N. Ishii, M. Fukushima, K. Sano, and K. Sawai. A study on dynamic behavior of a scroll compressor. In *1986 International Compressor Engineering Conference at Purdue University*, 1986.

N. Ishii, M. Yamamura, S. Muramatsu, S. Yamamoto, and M. Sakai. Mechanical Efficiency of a Variable Speed Scroll Compressor. In *1990 Purdue Compressor Conference*, pages 192–199, 1990.

N. Ishii, K. Bird, K. Sano, M. Oono, S. Iwamura, and T. Otokura. Refrigerant Leakage Flow Evaluation for scroll compressors. In *1996 Compressor Conference at Purdue University*, 1996a.

N. Ishii, M. Sakai, K. Sano, S. Yamamoto, and T. Otokura. A fundamental optimum design for high mechanical and volumetric efficiency of compact scroll compressors. In *1996 Compressor Conference at Purdue University*, 1996b.

N. Ishii, T. Oku, K. Anami, C. W. Knisely, K. Sawai, T. Morimoto, and K. Fujiuchi. Effects of Surface Roughness upon Gas Leakage Flow through small clearances in CO_2 scroll compressors. In *2008 International Compressor Engineering Conference at Purdue University*, 2008.

T. Itoh, M. Fujitani, and Y. Sakai. Leakage Characteristic of Scroll Compressor by Two Phase Flow Model in Consideration of Wall Oil Film Thickness [in Japanese]. *Japan Society of Mechanical Engineers*, 7(01-1391):158–165, 1990.

K. Jang and S. Jeong. Experimental investigation on convective heat transfer mechanism in a scroll compressor. *Int. J. Refrig.*, 29:744–753, 2006.

D. J. Kang, J. W. Kim, and C. B. Sohn. Effects of leakage flow model on the thermodynamic performance of a scroll compressor. In *2002 International Compressor Engineering Conference at Purdue University*, 2002.

G. Kemp, L. Elwood, and E. A. Groll. Evaluation of a Prototype Rotating Spool Compressor in Liquid Flooded Operation. In *20th International Compressor Engineering Conference at Purdue University*, 2010.

S. Klein. Engineering Equation Solver, 2010.

E. Kreyszig. *Advanced Engineering Mathematics (9th Edition)*. John Wiley and Sons, 2006.

J. E. Lawrie. *Glycerol and the Glycols*. American Chemical Society, 1928.

B.-C. Lee, T. Yanagisawa, M. Fukuta, and S. Choi. A study on the leakage characteristics of tip seal mechanism in the scroll compressor. In *2002 International Compressor Engineering Conference at Purdue University*, 2002.

Y.-R. Lee and W.-F. Wu. On the profile design of a scroll compressor. *Int. J. Refrig*, 18:308–317, 1995.

E. Lemmon. Pseudo-Pure Fluid Equations of State for the Refrigerant Blends R-410A, R-404A, R-507A, and R-407C. *International Journal of Thermophysics*, 24 (4):991–1006, 2003.

E. Lemmon, R. T. Jacobsen, S. G. Penoncello, and D. Friend. Thermodynamic Properties of Air and Mixtures of Nitrogen, Argon, and Oxygen from 60 to 2000 K at Pressures to 2000 MPa. *J. Phys. Chem. Ref. Data*, 29(3):331–385, 2000.

E. W. Lemmon and R. T. Jacobsen. Viscosity and Thermal Conductivity Equations for Nitrogen, Oxygen, Argon, and Air. *International Journal of Thermophysics*, 25: 21–69, 2004.

H. M. M. M. Lemmon, E.W. NIST Standard Reference Database 23: Reference Fluid Thermodynamic and Transport Properties-REFPROP, Version 9.0, 2010.

V. Lemort. *Contribution to the Characterization of Scroll Machines in Compressor and Expander Modes*. PhD thesis, University of Liège, 2008.

H. Li and L. Jin. Design Optimization of an oil-flooded Refrigeration single-screw compressor. In *2004 International Compressor Engineering Conference at Purdue University*, number C077, 2004.

H. Li and R. Wang. Performance improvement of r134a refrigerator compressor. In *2000 International Compressor Engineering Conference at Purdue University*, 2000.

H. Li, D. Wang, H. Wang, and P. Chen. Research of oil-injected scroll compressor working process. In *1992 Compressor Engineering Conference at Purdue University*, pages 118b1–118b13, 1992.

P. Liley and W. Gambill. *Chemical Engineering Handbook (5th Edition)*, chapter Physical and Chemical Data, pages 3–226 to 3–250. McGraw-Hill, 1973.

C. Lin, Y. Chang, K. Liang, and C. Hung. Temperature and thermal deformation analysis on scrolls of scroll compressor. *Applied Thermal Engineering*, 25:1724–1739, 2005.

Y. Liu, C. Hung, and Y. Chang. Design optimization of scroll compressor applied for frictional losses evaluation. *International Journal of Refrigeration*, 33(3):615 – 624, 2010. ISSN 0140-7007. doi: 10.1016/j.ijrefrig.2009.12.015.

V. P. Logvinyuk, V. V. Makarenkov, V. V. Malyshev, and G. M. Panchenkov. Solubility of gases in petroleum products. *Chemistry and Technology of Fuels and Oils*, 6(5):353–355, May 1970. doi: 10.1007/BF01171678.

G. Lorentzen. The use of natural refrigerants: a complete solution to the CFC/HCFC predicament. *International Journal of Refrigeration*, 18:190–197, 1995.

O. Lottin, P. Guillemet, and J.-M. Lebreton. Effects of synthetic oil in a compression refrigeration system using R410A. Part I: modelling of the whole system and analysis of its response to an increase in the amount of circulating oil. *International Journal of Refrigeration*, 26:772–782, 2003a.

O. Lottin, P. Guillemet, and J.-M. Lebreton. Effects of synthetic oil in a compression refrigeration system using R410A. Part II: quality of heat transfer and pressure losses within the heat exchangers. *Int. J. Refrig*, 26:783–794, 2003b.

D. L. Margolis, S. Craig, G. Nowakowski, and M. Inada. Modeling and simulation of a scroll compressor using bond graphs. In *1992 International Compressor Engineering Conference at Purdue University*, 1992.

K. N. Marsh, R. A. Perkins, and M. L. V. Ramires. Measurement and Correlation of the Thermal Conductivity of Propane from 86 K to 600 K at Pressures to 70 MPa. *J. Chem. Eng. Data*, 47:932–940, 2002.

W. L. Martz and A. M. Jacobi. Refrigerant-Oil Mixtures and Local Composition Modeling. Technical Report ACRC TR-68, University of Illinois, 1994.

W. L. Martz, C. M. Burton, and A. M. Jacobi. Local composition modelling of the thermodynamic properties of refrigerant and oil mixtures. *Int. J. Refrig*, 19:25–33, 1996.

W. McAdams, W. Wood, and R. Bryan. Vaporization inside Horizontal Tubes - II: Benzene-oil mixtures. *Transactions ASME*, 64:193, 1942.

Å. Melinder. *Properties of Secondary Working Fluids for Indirect Systems*. International Institute of Refrigeration, 2010.

Y. Mermond, M. Feidt, and C. Marvillet. Propriétés thermodynamiques et physiques des mélanges de fluides frigorigènes et d'huiles. *Int. J. Refrig*, 22:569–579, 1999.

H. Miyamoto and K. Watanabe. A Thermodynamic Property Model for Fluid-Phase Propane. *Int. J. Thermophys.*, 21:1045–1072, 2000.

M. J. Moran and H. N. Shapiro. *Fundamentals of Engineering Thermodynamics*. John Wiley and Sons, 2008.

E. Morishita and M. Sugihara. Scroll Compressor Analytical Model. In *1984 International Compressor Engineering Conference at Purdue University*, page 487, 1984.

S. Morris. Compressible gas-liquid flow through pipeline restrictions. *Chem. Eng. Process.*, 30:39–44, 1991.

A. Murta. General Polygon Clipper Library, 2010. URL `http://www.cs.man.ac.uk/~toby/alan/software/`.

H. K. Myong. Numerical investigation of fully developed turbulent fluid flow and heat transfer in a square duct. *International Journal of Heat and Fluid Flow*, 12(4): 344 – 352, 1991. ISSN 0142-727X. doi: 10.1016/0142-727X(91)90023-O.

National Renewable Energy Laboratories. Typical Meteorological Year 3 Dataset, 2005. URL `http://rredc.nrel.gov/solar/old_data/nsrdb/1991-2005/tmy3/`.

National Research Council (U.S.). Panel on Atmospheric Chemistry. *Halocarbons, effects on stratospheric ozone*. National Academy of Sciences, 1976. URL `http://books.google.com/books?id=a2YrAAAAYAAJ`.

T. Oku, K. Ishii, N. Ishii, C. W. Knisely, K. Sawai, T. Aya, and N. Ida. Leakage Tests of Wet CO_2 Gas with Oil-Mixture in Scroll Compressors and Its Use in Simulations of Optimal Performance. In *2006 Compressor Engineering Conference at Purdue University*, number C127, 2006.

K. T. Ooi and J. Zhu. Convective heat transfer in a scroll compressor chamber: a 2-D simulation. *International Journal of Thermal Sciences*, 43:677–688, 2004.

A. Pearson. *The Optimisation of Carbon Dioxide Refrigeration Systems*. PhD thesis, University of Strathclyde, 2005.

A. Pfenning. *VDI Warm Atlas*, chapter D5.1 Calculation of Vapor-Liquid Equilibria, pages 513–526. Springer Verlag, 2010. ISBN 3540778764.

R. Puff and M. Krueger. Influence of the main constructive parameters of a scroll compressor on its efficiency. In *1992 International Compressor Engineering Conference at Purdue University*, 1992.

Z. C. Qu and A. Tramschek. Investigation of a multistage Scroll Compressor with Oil-Injection. In *1996 International Compressor Engineering Conference at Purdue University*, 1996.

K. Rice. Oak Ridge National Labs Heat Pump Design Model Mark VI, 2005. URL `http://www.ornl.gov/~wlj/hpdm/MarkVI.shtml`.

A. Sakuda, K. Sawai, N. Iida, A. Hiwata, T. Morimoto, and N. Ishii. Performance improvement of scroll compressor with new sealing-oil supply mechanism. In *International Conference on Compressors and their Systems*, number C591/019, 2001.

K. Sawai, A. Hiwata, A. Sakuda, N. Iida, T. Morimoto, and N. Ishii. Experimental Study for High Efficiency on R410A Scroll Compressor - 2^{nd} Report: Efficiency Improvement with New Oil Injection to Compression Chambers [Japanese]. *Trans. of the JSRAE*, 26:387–395, 2009.

G. Scalabrin, P. Marchi, and F. Finezzo. A multiparameter thermal conductivity equation for R134a with an optimized functional form. *Fluid Phase Equilibria*, 245: 37–51, 2006a.

G. Scalabrin, P. Marchi, and R. Span. A Reference Multiparameter Viscosity Equation for R134a with an Optimized Functional Form. *J. Phys. Chem. Ref. Data*, 35: 839–868, 2006b.

G. Scalabrin, P. Marchi, and R. Span. A Reference Multiparameter Viscosity Equation for Propane with an Optimized Functional Form. *J. Phys. Chem. Ref. Data*, 35:1415–1442, 2006c.

C. Schein and R. Radermacher. Scroll compressor simulation model. *Journal of Engineering for Gas Turbines and Power*, 123:217–225, 2001. doi: 10.1115/1.1335483.

C. Seeton and P. Hrnjak. Thermophysical Properties of CO_2-Lubricant Mixtures and their affect on 2-Phase flow in small channels (less than 1mm). In *International Refrigeration and Air Conditioning Conference at Purdue University*, 2006.

C. Seeton, J. Fahl, and D. Henderson. Solubility, Viscosity, Boundary Lubrication and Miscibility of CO_2 and Synthetic Lubricants. In *2000 International Compressor Engineering Conference at Purdue University*, 2000.

P. J. Singh and G. C. Patel. A generalized performance computer program for oil flooded twin-screw compressors. In *1984 International Compressor Engineering Conference at Purdue University*, pages 544–553, 1984.

R. Span and W. Wagner. A New Equation of State for Carbon Dioxide Covering the Fluid Region from the Triple-Point Temperature to 1100 K at Pressures up to 800 MPa. *J. Phys. Chem. Ref. Data.*, 25:1509–1596, 1996.

R. Span, E. W. Lemmon, R. T. Jacobsen, W. Wagner, and A. Yokozeki. A Reference Equation of State for the Thermodynamic Properties of Nitrogen for Temperatures from 63.151 to 1000 K and Pressures to 2200 K. *J. Phys. Chem. Ref. Data*, 29: 1361–1433, 2000.

N. Stosic, A. Kovacevic, K. Hanjalic, and L. Milutinovic. Mathematical modelling of the oil influence upon the working cycle of screw compressors. In *1988 International Compressor Engineering Conference at Purdue University*, pages 354–361, 1988.

N. Stosic, L. Milutinovic, K. Hanjalic, and A. Kovacevic. Experimental investigation of the influence of oil injection upon the screw compressor working process. In *1990 International Compressor Engineering Conference at Purdue University*, pages 34–43, 1990.

K. Suefuji, M. Shiibayashi, and K. Tojo. Performance analysis of hermetic scroll compressors. In *1992 International Compressor Engineering Conference at Purdue University*, 1992.

S. Sunder and J. L. Smith Jr. Kissing heat transfer between the wraps of a scroll pump. *American Society of Mechanical Engineers, Advanced Energy Systems Division (Publication) AES*, 36:133 – 149, 1996.

S. Sunder and J. L. Smith Jr. Lumped parameter thermodynamic and heat transfer modeling of a scroll pump. In *American Society of Mechanical Engineers, Advanced Energy Systems Division*, volume 37, pages 417 – 427, Dallas, TX, USA, 1997.

N. Tagri and R. Jayaraman. Heat transfer studies on a spiral plate heat exchanger. *Trans. Inst. Chem. Eng.*, 40:161–168, 1962.

Y. Tang and J. S. Fleming. Simulation of the working process of an oil flooded helical screw compressor with liquid refrigerant injection. In *1992 International Compressor Engineering Conference at Purdue University*, pages 213–220, 1992.

C. Tegeler, R. Span, and W. Wagner. A New Equation of State for Argon Covering the Fluid Region for Temperatures From the Melting Line to 700 K at Pressures up to 1000 MPa. *J. Phys. Chem. Ref. Data*, 28:779–850, 1999.

M. Teodorescu, L. Lugo, and J. Fernández. Modeling of Gas Solubility Data for HFCs-Lubricant Oil Binary Systems by Means of the SRK Equation of State. *Int. J. Thermophys.*, 24:1043–1060, 2003.

J. Thome. *Engineering Data Book III.* Wolverine Tube, 2004.

R. Tillner-Roth and H. D. Baehr. A International Standard Formulation for the Thermodynamic Properties of 1,1,1,2-Tetrafluoroethane (HFC-134a) for Temperatures from 170 K to 455 K and Pressures up to 70 MPa. *J. Phys. Chem. Ref. Data*, 23:657–729, 1994.

R. Tillner-Roth, F. Harms-Watzenberg, and H. Baehr. Eine neue Fundamentalgleichung für Ammoniak (A new Equation of State for Ammonia). In *Deutscher Kälte- und Klimatechnischer Verein Tagung*, 1993.

K. Tojo, M. Ikegawa, N. Maeda, S. Machida, M. Shiibayashi, and N. Uchikawa. Computer modeling of scroll compressor with self adjusting back-pressure mechanism. In *1986 International Compressor Engineering Conference at Purdue University*, 1986.

G. Totten, S. Westbrook, and R. Shah. *Fuels and lubricants handbook: technology, properties, performance, and testing*. ASTM Intl, 2003.

C. Toublanc. *Amelioration Du Cycle Frigorifique Trans-Critique Au CO_2 Par Une Compression Refroidie: Evaluations Numerique et Experimentale [In French]*. PhD thesis, Le Conservatoire National Des Arts et Metiers, 2009.

C.-H. Tseng and Y.-C. Chang. Family design of scroll compressors with optimization. *Applied Thermal Engineering*, 26:1074–1086, 2006. doi: 10.1016/j.applthermaleng.2005.05.010.

R. Tufeu, D. Ivanov, Y. Garrabos, and B. L. Neindre. Thermal Conductivity of Ammonia in a Large Temperature and Pressure Range Including the Critical Region. *Bereicht der Bunsengesellschaft Phys. Chem.*, 88:422–427, 1984.

T. C. Wagner, A. A. Peracchio, M. E. Marler, and R. L. DeBlois. Heat transfer model for predicting metal temperatures in scroll compressor unit. *American Society of Mechanical Engineers, Advanced Energy Systems Division (Publication) AES*, 34: 131 – 141, 1995.

B. Wang, X. Li, and W. Shi. A general geometrical model of scroll compressors based on discretional initial angles of involute. *Int. J. Refrig.*, 28:958–966, 2005.

S. Wang, G. Chen, M. Fang, and Q. Wang. A new compressed air energy storage refrigeration system. *Energy Conversion and Management*, 47:3408–3416, 2006.

Z. Wang. A new type of curve used in the wrap design of the scroll compressor. In *1992 International Compressor Engineering Conference at Purdue University*, pages 1089–1098, 1992.

C. Wassgren. *ME 510 Gas Dynamics Class Notes (Purdue University)*. Boiler Copy Maker, 2009.

A. White and A. Meacock. An evaluation of the Effects of Water Injection on Compressor Performance. *Transactions of the ASME*, 126:748–754, 2004.

H. Wu, Z. Xing, and P. Shu. Theoretical and experimental study on indicator diagram of twin screw refrigeration compressor. *Int. J. Refrig.*, 27:331–338, 2004.

J. Wu and G. Jin. The computer simulation of oil-flooded single screw compressors. In *1988 International Compressor Engineering Conference at Purdue University*, pages 362–368, 1988.

D. Xin, J. Feng, X. Jia, and X. Peng. An investigation into oil-gas two-phase leakage flow through micro gaps in oil-injected compressors. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 224 (4):925 – 933, 2010. doi: 10.1243/09544062JMES1704.

T. Yanagisawa and T. Shimizu. Leakage losses with a rolling piston type rotary compressor. i. radial clearance on the rolling piston. *International Journal of Refrigeration*, 8(2):75 – 84, 1985a. ISSN 0140-7007. doi: 10.1016/0140-7007(85)90077-5.

T. Yanagisawa and T. Shimizu. Leakage losses with a rolling piston type rotary compressor. II. Leakage losses through clearances on rolling piston faces. *International Journal of Refrigeration*, 8(3):152 – 158, 1985b. ISSN 0140-7007. doi: 10.1016/0140-7007(85)90155-0.

T. Yanagisawa and T. Shimizu. Friction losses in rolling piston type rotary compressors. III. *International Journal of Refrigeration*, 8(3):159 – 165, 1985c. ISSN 0140-7007. doi: 10.1016/0140-7007(85)90156-2.

T. Yanagisawa, M. C. Cheng, M. Fukuta, and T. Shimizu. Optimum operating pressure ratio for scroll compressor. In *1990 International Compressor Engineering Conference at Purdue University*, pages 425–433, 1990.

Q. Yang, L. Li, Y. Zhao, and P. Shu. Experimental measurement of axial clearance in scroll compressor using eddy current displacement sensor. *Proc. IMechE Part C: J. Mechanical Engineering Science*, 222:1315–1320, 2008.

C. Yaws, editor. *Chemical Properties Handbook*. McGraw-Hill, 1999.

A. Yokozeki. Solubility of Refrigerants in Various Lubricants. *Int. J. Thermophys.*, 22:1057–1071, 2001.

A. Yokozeki. Time-dependent behavior of gas absorption in lubricant oil. *Int. J. Refrig*, 25:695–704, 2002.

A. Yokozeki. Theoretical performances of various refrigerant-absorbent pairs in a vapor-absorption refrigeration cycle by the use of equations of state. *Applied Energy*, 80:383–399, 2005.

A. Yokozeki. Solubility correlation and phase behaviors of carbon dioxide and lubricant oil mixtures. *Applied Energy*, 84:159–175, 2007.

M. Youbi-Idrissi and J. Bonjour. The effect of oil in refrigeration: Current research issues and critical review of thermodynamic aspects. *Int. J. Refrig*, 31:165–179, 2008.

M. Youbi-Idrissi, J. Bonjour, C. Marvillet, and F. Meunier. Impact of refrigerant-oil solubility on an evaporator performances working with R-407C. *Int. J. Refrig*, 26: 284–292, 2003.

M. Youbi-Idrissi, J. Bonjour, M.-F. Terrier, C. Marvillet, and F. Meunier. Oil presence in an evaporator: experimental validation of a refrigerant/oil mixture enthalpy calculation model. *Int. J. Refrig*, 27:215–224, 2004.

Y. Youn, N.-K. Cho, B.-C. Lee, and M.-K. Min. The characteristics of tip leakage in scroll compressors for air conditioners. In *2000 International Compressor Engineering Conference at Purdue University*, 2000.

X. Yuan, Z. Chen, and Z. Fan. Calculating model and experimental investigation of gas leakage. In *1992 International Compressor Engineering Conference at Purdue University*, 1992.

Q. Zheng, Y. Sun, S. Li, and Y. Wang. Thermodynamic Analyses of Wet Compression Process in the Compressor of Gas Turbine. *Journal of Turbomachinery*, 125: 489–496, 2003.

J. Zhu and K. Ooi. Prediction of Flow and Heat Transfer Within a Compressor Working Chamber. In *Proceedings of the ASME Advanced Energy Systems Division*, pages 463–468, 1997.

S. Zivi. Estimation of Steady-State Void-Fraction by Means of the Principle of Minimum Entropy Production. *Journal of Heat Transfer*, 86:247–252, 1964.

THEORETICAL AND EXPERIMENTAL ANALYSIS OF LIQUID FLOODED COMPRESSION IN SCROLL COMPRESSORS

VOLUME 2

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Ian Hadley Bell

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

May 2011

Purdue University

West Lafayette, Indiana

APPENDICES

Appendix A: Code For Flooded Cycle Analysis

```
## ######################################################
##             FloodedCycle.py
## ######################################################
import sys,os
import matplotlib
import pylab,copy
import matplotlib.transforms as mtransforms

params = {'axes.labelsize': 10,
          'text.fontsize': 8,
          'legend.fontsize': 8,
          'legend.labelspacing':0.2,
          'xtick.labelsize': 10,
          'ytick.labelsize': 10,
          'lines.linewidth': 0.5,
          'font.family' : 'Times New Roman',
          'text.usetex': False}
pylab.rcParams.update(params)

from FloodedCycleComponents import (Compressor, Condenser,
    Evaporator,
     EvaporatorVals,CompressorVals,CondenserVals, Separator,
         SeparatorVals,
     OilCooler, OilCoolerVals,Regenerator,RegeneratorVals,
         OilExpansion,
     OilExpansionVals)
from FloodedCycleProps import Q_Th, Density_mix,
    Solubility_Ref_in_Liq
from CoolProp.CoolProp import Props,PrintSaturationTable,Tcrit,pcrit
from FloodedCyclePlots import OverlayPH, OverlayTs, BaselinePH,
    BaselineTs
from numpy import linspace, meshgrid, array, loadtxt, size, shape
import numpy as np
import scipy
from scipy.optimize import fsolve, fminbound, fmin_cg, fmin_bfgs,
    fmin,fmin_tnc
from math import floor

class CycleInputVals:
    def __init__(self):
        vars=['Ref','Liq','DT_pinch' 'DT_subcool','DT_superheat',
            'Cycle','m_dot_g','T_L','T_H']
        for f in vars:
            self.__dict__.update(f=None)
        self.m_dot_g = 1.0
        self.Vdot = 20e-6*60
        self.DT_pinch = 5.0
        self.DT_subcool = 7.0
        self.DT_superheat = 5.0
        self.eta_comp = 0.7
        self.eta_m = 0.0
        self.effReg = 0.9
        self.y_sep = 0.0
    def TableDefaults(self,fName='DefaultParams.tex'):
        """
        Writes the default parameters to a LaTeX Table with filename
            fName
        """
        f=open(fName,'w')
        head='%Made from FloodedCycle.py function CycleInputVals.
            TableDefaults()\n'
        head=head+r'\begin{table*}[ht]'+'\n'
        head=head+r'\setlength{\tabcolsep}{3pt}'+'\n'
        head=head+r'\renewcommand*\arraystretch{1.0}'+'\n'
        head=head+r'\centering'+'\n'
        head=head+r'\caption{Default parameters for flooded cycle
            modeling.}'+'\n'
```

```
        head=head+r'\label{tab:DefaultParameters}'+'\n'
        head=head+r'\begin{tabular}{*{%d}{c}}' % 9 +'\n'
        head=head+r'\hline\hline' +'\n'
        head=head+r'Parameter & Value'+ r'\\'+'\hline\n'
        f.write(head)
        f.write(r'$\dot V_{comp}$ [m$^3$/h]'+' & '+"%0.2f" %(self.
            Vdot*3600.0,)+r'\\'+'\n')
        f.write(r'$\Delta T_{pinch}$ [K]'+' & '+"%0.1f" %(self.
            DT_pinch,)+r'\\'+'\n')
        f.write(r'$\Delta T_{subcool}$ [K]'+' & '+"%0.1f" %(self.
            DT_subcool,)+r'\\'+'\n')
        f.write(r'$\Delta T_{superheat}$ [K]'+' & '+"%0.1f" %(self.
            DT_superheat,)+r'\\'+'\n')
        f.write(r'$\eta_{comp}$ [-]'+' & '+"%0.1f" %(self.eta_comp,)
            +r'\\'+'\n')
        f.write(r'$\eta_{hyd-exp}$ [-]'+' & '+"%0.1f" %(self.eta_m,)
            +r'\\'+'\n')
        f.write(r'$\varepsilon_{Reg}$ [-]'+' & '+"%0.1f" %(self.
            effReg,)+r'\\'+'\n')
        f.write(r'$x_{g,s}$ [-]'+' & '+"%0.1f" %(self.y_sep,)+r'\\'+
            '\n')
        f.write(r'\hline\hline'+'\n')
        f.write(r'\end{tabular}'+'\n')
        f.write(r'\end{table*}'+'\n')
        f.close()
class PerformanceVals:
    def __init__(self):
        pass
class BaseVals:
    def __init__(self, **kwds):
        self.__dict__['Evaporator'] = EvaporatorVals()
        self.__dict__['Compressor'] = CompressorVals()
        self.__dict__['Condenser'] = CondenserVals()
        self.__dict__['Performance'] = PerformanceVals()
class FloodVals:
    def __init__(self, **kwds):
        self.__dict__['Evaporator'] = EvaporatorVals()
        self.__dict__['Compressor'] = CompressorVals()
        self.__dict__['Condenser'] = CondenserVals()
        self.__dict__['Separator'] = SeparatorVals()
        self.__dict__['OilCooler'] = OilCoolerVals()
        self.__dict__['OilExpansion'] = OilExpansionVals()
        self.__dict__['Regenerator'] = RegeneratorVals()
        self.__dict__['Performance'] = PerformanceVals()
def BaselineCycle(CycleInputs):
    #Instantiate the base class (data structure)
    Base=BaseVals()
    #Pull values
    Ref =           CycleInputs.Ref
    Liq =           CycleInputs.Liq
    m_dot_g =       CycleInputs.m_dot_g
    DT_pinch =      CycleInputs.DT_pinch
    DT_subcool =    CycleInputs.DT_subcool
    DT_superheat =  CycleInputs.DT_superheat
    eta_comp =      CycleInputs.eta_comp
    T_L =           CycleInputs.T_L
    T_H =           CycleInputs.T_H
    #Determine temperatures
    T_evap_out=T_L-DT_pinch;
    T_evap=T_L-DT_superheat-DT_pinch;
    T_cond=T_H+DT_subcool+DT_pinch;
    T_cond_out=T_H+DT_subcool;
    #Evaporation and condensing pressures
    p_evap=Props('P','T',T_evap,'Q',1,Ref);
```

```
if CycleInputs.Cycle == 'Subcritical':
    p_cond=Props('P','T',T_cond,'Q',1.0,Ref)
elif CycleInputs.Cycle == 'Transcritical':
    p_cond=CycleInputs.p_cond
else:
    print "Invalid Cycle type"
if CycleInputs.Vdot != None:
    f=lambda mdot_total: (mdot_total /
        Density_mix(Ref,Liq,T_evap_out,p_evap,0.0))-CycleInputs.
            Vdot
    m_dot_g=fsolve(f,0.1)
# Inputs for Compressor model
Base.Compressor.T_in = T_evap_out
Base.Compressor.p_in = p_evap
Base.Compressor.p_out = p_cond
Base.Compressor.xL = 0.0 #no flooding
Base.Compressor.eta_co = eta_comp
Base.Compressor.mdot_total = m_dot_g #since there is no flooding
# Compressor Model
Compressor(Ref,Liq,Base.Compressor)
# Inputs for Condenser model
kwargs={'T_in' : Base.Compressor.T_out,
        'p_in' : Base.Compressor.p_out,
        'T_H' : T_H,
        'DT_subcool' : DT_subcool,
        'DT_pinch' : DT_pinch,
        'mdot' : m_dot_g,
        'h_in' : Base.Compressor.h_out,
        'Type' : CycleInputs.Cycle}
Base.Condenser.update(**kwargs)
# Condenser Model
Condenser(Ref,Liq,Base.Condenser)
# Quality at the inlet to the evaporator determined as a
# function of enthalpy and saturation temperature
s1=Props('S','T',T_evap,'Q',0,'R744')
s2=Props('S','T',T_evap,'Q',1,'R744')
s3=Props('S','T',T_evap,'Q',0.43,Ref)
rho3=Props('D','T',T_evap,'Q',0.43,Ref)
x_in=Q_Th(Ref,Liq,T_evap,Base.Condenser.h_out,0.3)
# Inputs for Evaporator model
kwargs={'T_out' : T_evap_out,
        'p_out' : p_evap,
        'h_in' : Base.Condenser.h_out,
        's_in' : Props('S','T',T_evap,'Q',x_in,Ref),
        'mdot' : m_dot_g,
        'T_in' : T_evap,
        'T_L' : T_L}
Base.Evaporator.update(**kwargs)
# Evaporator Model
Evaporator(Ref,Liq,Base.Evaporator)
#Cycle Performance Metrics for AC Mode
Capacity = Base.Evaporator.Q
NetPower = Base.Compressor.Wdot
COP = Capacity/NetPower
COPCarnot = T_L/(T_H-T_L)
etaII=COP/COPCarnot
Base.Performance.Capacity=Capacity
Base.Performance.NetPower=NetPower
Base.Performance.COP=COP
Base.Performance.COPCarnot=COPCarnot
Base.Performance.etaII=etaII
#Cycle Performance Metrics for HP Mode
Capacity_HP = Base.Condenser.Q
```

```
    NetPower_HP = Base.Compressor.Wdot
    COP_HP = -Capacity_HP/NetPower_HP
    COPCarnot_HP = T_H/(T_H-T_L)
    etaII_HP=COP_HP/COPCarnot_HP
    Base.Performance.Capacity_HP=Capacity_HP
    Base.Performance.NetPower_HP=NetPower_HP
    Base.Performance.COP_HP=COP_HP
    Base.Performance.COPCarnot_HP=COPCarnot_HP
    Base.Performance.etaII_HP=etaII_HP
    return Base
def FloodedCycle(CycleInputs):
    # Instantiate the base class (data structure)
    Flood=FloodVals()
    # Pull values common to flooded and baseline systems
    Ref =           CycleInputs.Ref
    Liq =           CycleInputs.Liq
    m_dot_g =       float(CycleInputs.m_dot_g)
    eta_comp =      float(CycleInputs.eta_comp)
    DT_pinch =      float(CycleInputs.DT_pinch)
    DT_subcool =    float(CycleInputs.DT_subcool)
    DT_superheat =  float(CycleInputs.DT_superheat)
    T_L =           float(CycleInputs.T_L)
    T_H =           float(CycleInputs.T_H)
    # Parameters only for flooded system
    T_comp_guess =  float(CycleInputs.T_comp_in) # guess of inlet
       temp to compressor
    xL =            float(CycleInputs.xL)
    effReg =        float(CycleInputs.effReg)
    y_sep =         float(CycleInputs.y_sep)
    # Determine temperatures
    T_evap_out=T_L-DT_pinch;
    T_evap=T_L-DT_superheat-DT_pinch;
    T_cond=T_H+DT_subcool+DT_pinch;
    T_cond_out=T_H+DT_subcool;
    # Evaporation and condensing pressures
    p_evap=Props('P','T',T_evap,'Q',1,Ref);
    if CycleInputs.Cycle == 'Subcritical':
        p_cond=Props('P','T',T_cond,'Q',1,Ref)
    elif CycleInputs.Cycle == 'Transcritical':
        p_cond=CycleInputs.p_cond
    else:
        print "Invalid Cycle type"
    # Flow rates of total and flooding liquid as a function of oil
       mass frac.
    m_dot_total=0.0
    m_dot_liq=0.0
    iter=1
    r=999
    eps=1e-3
    while abs(r)>eps:
        if (iter==1):
            x1=float(T_comp_guess)
            T_comp_in=x1
        if (iter==2):
            x2=float(T_comp_guess)+0.1
            T_comp_in=x2
        if (iter>2):
            T_comp_in=x2
        if CycleInputs.Vdot == None:
            # Flow rates of total and flooding liquid as a function
               of oil mass frac.
            m_dot_total=m_dot_g*(1.0+xL/(1.0-xL))
            m_dot_liq=m_dot_g*xL/(1.0-xL)
        else:
```

```
    m_dot_total=Density_mix(Ref,Liq,T_comp_in,p_evap,xL)*
        CycleInputs.Vdot
    m_dot_g=(1-xL)*m_dot_total
    m_dot_liq=xL*m_dot_total
#Inputs for Compressor Model
kwargs={'T_in' : T_comp_in,
        'p_in' : p_evap,
        'p_out' : p_cond,
        'xL' : xL,
        'eta_co' : eta_comp,
        'mdot_total' : m_dot_total
        }
Flood.Compressor.update(**kwargs)
# Compressor Model
Compressor(Ref,Liq,Flood.Compressor)
#Inputs for Separator Model
kwargs={'T_in' : Flood.Compressor.T_out,
        'p_in' : p_cond,
        'mdot_g' : m_dot_g,
        'mdot_L' : m_dot_liq,
        'y_out' : y_sep
        }
Flood.Separator.update(**kwargs)
Solubility_error=False
if CycleInputs.y_sep==-1.0 and Flood.Compressor.xL > 1e-6:
    (Flood.Separator.y_out,Solubility_error)=
        Solubility_Ref_in_Liq(Ref,Liq,Flood.Compressor.T_out,
        p_cond)
else:
    Flood.Separator.y_out=0.0
if Solubility_error==True:
    print "T: %0.1f p %0.1f" %(Flood.Compressor.T_out,p_cond
        )
# Separator Model
Separator(Flood.Separator)
#Inputs for Oil Cooler Model
kwargs={'T_in' : Flood.Separator.T_g,
        'p_in' : Flood.Separator.p_g,
        'mdot_g_s' : Flood.Separator.mdot_g_s,
        'mdot_L' : m_dot_liq,
        'DT_pinch' : DT_pinch,
        'DT_subcool' : DT_subcool,
        'T_H' : T_H,
        'Type' : 'Subcritical'
        }
Flood.OilCooler.update(**kwargs)
# Oil Cooler Model
OilCooler(Ref,Liq,Flood.OilCooler)
# Inputs for Condenser model
kwargs={'T_in' : Flood.Compressor.T_out,
        'p_in' : Flood.Compressor.p_out,
        'T_H' : T_H,
        'DT_subcool' : DT_subcool,
        'DT_pinch' : DT_pinch,
        'mdot' : Flood.Separator.mdot_g_out,
        'h_in' : Flood.Compressor.h_out,
        'Type' : CycleInputs.Cycle}
Flood.Condenser.update(**kwargs)
# Condenser Model
Condenser(Ref,Liq,Flood.Condenser)
# Inputs for Regenerator model
kwargs={'Thi': Flood.Condenser.T_out,
        'phi': p_cond,
```

```
            'Tci': T_evap_out,
            'pci': p_evap,
            'epsilon' : CycleInputs.effReg,
            'mdot' : Flood.Separator.mdot_g_out}
    Flood.Regenerator.update(**kwargs)
    # Regenerator Model
    Regenerator(Ref,Liq,Flood.Regenerator)
    # Quality at the inlet to the evaporator determined as a
    # function of enthalpy and saturation temperature
    x_in=Q_Th(Ref,Liq,T_evap,Flood.Regenerator.hho,0.3)
    # Inputs for Evaporator model
    kwargs={'T_out' : T_evap_out,
            'p_out' : p_evap,
            'h_in' : Flood.Regenerator.hho,
            's_in' : Props('S','T',T_evap,'Q',x_in,Ref),
            'mdot' : Flood.Separator.mdot_g_out,
            'T_in' : T_evap,
            'T_L' : T_L}
    Flood.Evaporator.update(**kwargs)
    # Evaporator Model
    Evaporator(Ref,Liq,Flood.Evaporator)
    # Inputs for Oil Expansion device model
    kwargs={'T_in': T_cond_out,
            'T_evap': T_evap,
            'p_in': p_cond,
            'p_out': p_evap,
            'mdot_ref' : Flood.Separator.mdot_g_s,
            'mdot_liq' :  m_dot_liq,
            'eta_m' : CycleInputs.eta_m}
    Flood.OilExpansion.update(**kwargs)
    # Regenerator Model
    OilExpansion(Ref,Liq,Flood.OilExpansion)
    # Take residual on mixing point
    r_mix=(-Flood.Compressor.mdot_total*Flood.Compressor.h_in +
        Flood.OilCooler.mdot_total*Flood.OilCooler.h_out+
        Flood.Separator.mdot_g_out*Flood.Regenerator.hco)
    r=(Flood.Compressor.Wdot+Flood.Evaporator.Q+Flood.Condenser.
       Q+Flood.OilCooler.Q+Flood.OilExpansion.Wdot)
    if (iter==1):
        y1=r
    if (iter>1):
        y2=r
        x3=x2-y2/(y2-y1)*(x2-x1)
        change=abs(y2/(y2-y1)*(x2-x1))
        y1=y2
        x1=x2
        x2=x3
    iter=iter+1
#Cycle Performance Metrics in AC Mode
Capacity = Flood.Evaporator.Q
NetPower = Flood.Compressor.Wdot+Flood.OilExpansion.Wdot
COP = Capacity/NetPower
COPCarnot = T_L/(T_H-T_L)
etaII=COP/COPCarnot
Flood.Performance.Capacity=Capacity
Flood.Performance.NetPower=NetPower
Flood.Performance.COP=COP
Flood.Performance.COPCarnot=COPCarnot
Flood.Performance.etaII=etaII
Capacity_HP = Flood.Condenser.Q+Flood.OilCooler.Q
NetPower_HP = Flood.Compressor.Wdot+Flood.OilExpansion.Wdot
COP_HP = -Capacity_HP/NetPower_HP
```

```
    COPCarnot_HP = T_H/(T_H-T_L)
    etaII_HP=COP_HP/COPCarnot_HP
    Flood.Performance.Capacity_HP=Capacity_HP
    Flood.Performance.NetPower_HP=NetPower_HP
    Flood.Performance.COP_HP=COP_HP
    Flood.Performance.COPCarnot_HP=COPCarnot_HP
    Flood.Performance.etaII_HP=etaII_HP
    return Flood
def Optimize_Flooded_COP(CycleInputs,Mode='AC',**kwargs):
    """
    Optimizes the COP of the flooded system.
    For transcritical systems, optimizes both oil mass fraction
    and gas cooler pressure
    """
    if 'factr' in kwargs:
        factr=kwargs['factr']
    else:
        factr=1000
    Flood=FloodVals()
    # Objective function for transcritical systems as
    # function of xL and gas cooler pressure
    def OBJECTIVE_transcrit_flooded(x):
        CycleInputs.Cycle='Transcritical'
        CycleInputs.xL = float(x[0])
        CycleInputs.p_cond = float(x[1])
        CycleInputs.T_comp_in = CycleInputs.xL*CycleInputs.T_H+(1.0-
            CycleInputs.xL)*CycleInputs.T_L
        Flood_max=FloodedCycle(CycleInputs)
        if Mode=='AC':
            return (1/Flood_max.Performance.COP)
        elif Mode=='HP':
            return (1/Flood_max.Performance.COP_HP)
    #Objective function for subcritical system as function of xL
    def OBJECTIVE_subcrit_flooded(x):
        CycleInputs.Cycle='Subcritical'
        CycleInputs.xL = float(x)
        CycleInputs.T_comp_in = CycleInputs.xL*CycleInputs.T_H+(1.0-
            CycleInputs.xL)*CycleInputs.T_L
        Flood_max=FloodedCycle(CycleInputs)
        if Mode=='AC':
            return (1/Flood_max.Performance.COP)
        elif Mode=='HP':
            return (1/Flood_max.Performance.COP_HP)
    if (CycleInputs.T_H+CycleInputs.DT_pinch+CycleInputs.DT_subcool)
        >Tcrit(CycleInputs.Ref):
        # Cycle is transcritical
        b=[(0.0,0.9),(7380.0,16000.0)]
        (x,nfeval,rc)=scipy.optimize.fmin_l_bfgs_b(
            OBJECTIVE_transcrit_flooded,array([0.5,12500.0]),
            approx_grad=True,bounds=b,factr=factr)
#        (x,nfeval,rc)=scipy.optimize.fmin_tnc(
    OBJECTIVE_transcrit_flooded,array([0.5,12500.0]),approx_grad=True
    ,bounds=b)
        xL=x[0]
        p=x[1]
        CycleInputs.T_comp_in = CycleInputs.xL*CycleInputs.T_H+(1.0-
            CycleInputs.xL)*CycleInputs.T_L
        Flood=FloodedCycle(CycleInputs)
    else:
        # Cycle is subcritical
        xL=fminbound(OBJECTIVE_subcrit_flooded,0.0,0.9)
        CycleInputs.xL=float(xL)
        CycleInputs.T_comp_in = CycleInputs.xL*CycleInputs.T_H+(1.0-
            CycleInputs.xL)*CycleInputs.T_L
        Flood=FloodedCycle(CycleInputs)
```

```
    return Flood
def Optimize_Baseline_COP(CycleInputs,Mode='AC'):
    """
    Optimizes the COP of the baseline system.
    For transcritical systems, optimizes gas cooler pressure,
    For subcritical systems, no optimization
    """
    Base=BaseVals()
    # Objective function for transcritical systems as
    # function of gas cooler pressure
    def OBJECTIVE_transcrit_dry(p):
        CycleInputs.Cycle='Transcritical'
        CycleInputs.xL = 0.0000001
        CycleInputs.p_cond = float(p)
        CycleInputs.T_comp_in = CycleInputs.xL*CycleInputs.T_H+(1.0-
            CycleInputs.xL)*CycleInputs.T_L
        Flood_max=FloodedCycle(CycleInputs)
        if Mode=='AC':
            return (1/Flood_max.Performance.COP)
        elif Mode=='HP':
            return (1/Flood_max.Performance.COP_HP)
    if (CycleInputs.T_H+CycleInputs.DT_pinch+CycleInputs.DT_subcool)
        >Tcrit(CycleInputs.Ref):
        # Transcritical, optimize the gas cooler pressure
        CycleInputs.p_cond=float(fminbound(OBJECTIVE_transcrit_dry,
            pcrit(CycleInputs.Ref),pcrit(CycleInputs.Ref)*2.0))
        return BaselineCycle(CycleInputs)
    else:
        # Subcritical, no optimization required
        CycleInputs.Cycle='Subcritical'
        return BaselineCycle(CycleInputs)
if __name__== "__main__":
    pass

## ####################################################
##           FloodedCycleComponents.py
## ####################################################
from CoolProp.CoolProp import Props
from FloodedCycleProps import *
class CompressorVals:
    def __init__(self):
        vars=['T_in','p_in','p_out','xL','eta_co','mdot_total','
            T_out','h_in',
        'h_out','s_in','s_out','Wdot','Edot','Q']
        for f in vars:
            self.__dict__[f] = None
    def update(self,**kwargs):
        for f in kwargs:
            if f in self.__dict__:
                self.__dict__[f] = kwargs[f]
            else:
                print "Field "+f+" not found"
def Compressor(Ref,Liq,Comp):
        #"State 1 is the inlet state to the compressor"
        #"State 2 is the outlet state to the compressor"
    #Retrieve values
    T1=Comp.T_in
    P1=Comp.p_in
    P2=Comp.p_out
    xL=Comp.xL
    eta_c=Comp.eta_co
    mdot_total=Comp.mdot_total
    T0=298
    xL2 = xL
    s1 = Entropy_mix(Ref, Liq, T1, P1, xL)
```

```
    h1 = Enthalpy_mix(Ref, Liq, T1, P1, xL)
    T2s = T_sp(Ref, Liq, s1, P2, xL, T1+50)
    h2sm = Enthalpy_mix(Ref, Liq, T2s, P2, xL)
    h2 = (h2sm-h1)/eta_c+h1
    T2 = T_hp(Ref, Liq, h2, P2, xL, T2s)
    s2 = Entropy_mix(Ref, Liq, T2, P2, xL)
    Wdot = mdot_total*(h2-h1)                                    #
       [W
    Edot=T0*mdot_total*(s2-s1)                                   #
       [kW]
    Q=0.0
    #Store values
    Comp.T_out=T2
    Comp.s_in=s1
    Comp.h_out=h2
    Comp.h_in=h1
    Comp.s_out=s2
    Comp.Wdot=Wdot
    Comp.Q=Q
    Comp.Edot=Edot
class CondenserVals:
    def __init__(self):
        fields=['T_in','p_in','h_in','T_H','DT_subcool','DT_pinch',
        'mdot','h_out','T_out','p_out','Q','Edot','Type']
        for f in fields:
            self.__dict__[f] = None
    def update(self,**kwargs):
        for f in kwargs:
            if f in self.__dict__:
                self.__dict__[f] = kwargs[f]
            else:
                print "Field "+f+" not found"
def Condenser(Ref,Liq,Cond):
    # For now, assume that all the heat transfer occurs in the gas
       phase as
    # either condensation of gas or supercritical heat rejection"
    T1 = Cond.T_in
    P1 = Cond.p_in
    DT_sub = Cond.DT_subcool
    DT_pinch = Cond.DT_pinch
    T_H = Cond.T_H
    mdot = Cond.mdot
    Type = Cond.Type
    T0 = 298
    P2 = P1
    h1 = Enthalpy_mix(Ref,Liq,T1,P1,0)
    s1 = Entropy_mix(Ref,Liq,T1,P1,0)
    if (Type == "Subcritical"):
        # "Condensation of Gas and Cooling of Liquid"
        T_cond = T_H+DT_sub+DT_pinch
        T_cond_out = T_H+DT_pinch
        T2 = T_cond_out
        h2 = Enthalpy_mix(Ref,Liq,T_cond_out,P1,0)
        s2 = Entropy_mix(Ref,Liq,T_cond_out,P1,0)
    else:
        # "Supercritical Heat Rejection"
        T_cond = T_H+DT_pinch # "not really a condensation
           temperature..."
        T2 = T_cond
        h2 = Enthalpy_mix(Ref,Liq,T_cond,P1,0)
        s2 = Entropy_mix(Ref,Liq,T_cond,P1,0)
    Qh = -mdot * (h1-h2)
    Edot=T0*(mdot*(s2-s1)-Qh/T_H)
    Cond.h_out=h2
    Cond.T_out=T2
    Cond.s_in=s1
```

```
    Cond.s_out=s2
    Cond.p_out=P2
    Cond.Q=Qh
    Cond.Edot=Edot
    Cond.h_in=h1
    Cond.Tsat=T_cond
class SeparatorVals:
    def __init__(self):
        fields=['T_in','p_in','mdot_g','mdot_g_s',
        'T_out_L','p_out_L','y_out','mdot_L']
        for f in fields:
            self.__dict__[f] = None
    def update(self,**kwargs):
        for f in kwargs:
            if f in self.__dict__:
                self.__dict__[f] = kwargs[f]
            else:
                print "Field "+f+" not found"
def Separator(Sep):
    """
    In the separator it is assumed that all the liquid is
    fully separated and passes through in the liquid phase.
    There is no pressure drop, but the ficticious change in
    solubility results in some of the refrigerant now going
    with the oil in solution rather than staying in the vapor
    phase. The total mass flow rate of refrigerant entering the
    separator is given and expressed as:
    mdot_g = mdot_g_vapor + mdot_g_solved
    The oil mass fraction is defined by
    xL=mdot_oil / (mdot_g+mdot_oil)
    If the inlet/outlet equilibrium solubility refrigerant mass
    fractions are y_in and y_out, the oil mass fraction at the inlet
        is
    """
    p =         Sep.p_in
    T =         Sep.T_in
    mdot_g_in = Sep.mdot_g # Total refrigerant mass flow rate (
        solved + vapor)
    mdot_L =    Sep.mdot_L # Liquid mass flow rate
    # Refrigerant fraction in oil in the range [0,1] though
        generally
    # less than 0.5 (50%)
    y_out =  Sep.y_out
    # Amount of refrigerant solved in oil at outlet
    mdot_g_s_out=mdot_L *y_out / (1-y_out)
    # Remaining refrigerant goes as vapor goes to the condenser
    mdot_g_vap_out=mdot_g_in-mdot_g_s_out
    #Store values
    Sep.T_L = T
    Sep.p_L = p
    Sep.xL_L = 1.0-y_out # from xL=mL/(mL*(1+y))
    Sep.mdot_g_s=mdot_g_s_out
    Sep.T_g = T
    Sep.p_g = p
    Sep.xL_g = 0
    Sep.mdot_g_out = mdot_g_vap_out
class OilCoolerVals:
    def __init__(self):
        fields=['T_in','T_H','p_in','DT_pinch','mdot_g_s','mdot_L','
            DT_pinch','DT_subcool','Type']
        for f in fields:
            self.__dict__[f] = None
    def update(self,**kwargs):
```

```
        for f in kwargs:
            if f in self.__dict__:
                self.__dict__[f] = kwargs[f]
            else:
                print "Field "+f+" not found"
def OilCooler(Ref,Liq,OilCooler):
    """
    Oil is cooled to slightly above the ambient temperature,
    and the solved refrigerant is condensed back to liquid and
    subcooled by the same amount as the primary refrigerant path
    """
    # Retrieve values from class
    T1=OilCooler.T_in
    P1=OilCooler.p_in
    DT_pinch=OilCooler.DT_pinch
    DT_subcool=OilCooler.DT_subcool
    mdot_oil=OilCooler.mdot_L
    mdot_ref=OilCooler.mdot_g_s #refrigerant vapor solved in oil
    T_H=OilCooler.T_H
    Type=OilCooler.Type

    P2 = P1
    T0 = 298
    xL=mdot_oil/(mdot_oil+mdot_ref)
    #### Oil cooler
    T2 = T_H+DT_pinch
    T_avg = (T1+T2)/2
    cp = cp_mix(Ref,Liq,T_avg,P1,1)
    Q_oil=-mdot_oil*cp*(T1-T2)
    s1 = Entropy_mix(Ref,Liq,T1,P1,1)
    s2 = Entropy_mix(Ref,Liq,T2,P2,1)
    OilCooler.Edot_oil=T0*(mdot_oil*(s2-s1)-Q_oil/T_H);
    OilCooler.Q_oil=Q_oil
    h2_oil=Enthalpy_mix(Ref,Liq,T2,P2,1)
    #### Refrigerant condenser
    h1 = Enthalpy_mix(Ref,Liq,T1,P1,0)
    s1 = Entropy_mix(Ref,Liq,T1,P1,0)
    if (Type == "Subcritical"):
        # "Condensation of Gas and Cooling of Liquid"
        T_cond = T_H+DT_subcool+DT_pinch
        h2 = Enthalpy_mix(Ref,Liq,T2,P1,0)
        s2 = Entropy_mix(Ref,Liq,T2,P1,0)
    else:
        # "Supercritical Heat Rejection"
        h2 = Enthalpy_mix(Ref,Liq,T2,P1,0)
        s2 = Entropy_mix(Ref,Liq,T2,P1,0)
    Q_ref = -mdot_ref * (h1-h2)
    h2_ref=h2
    OilCooler.Edot_ref=T0*(mdot_ref*(s2-s1)-Q_ref/T_H)
    OilCooler.Q_ref=Q_ref
    OilCooler.Edot=OilCooler.Edot_ref+OilCooler.Edot_oil
    OilCooler.Q=Q_ref+Q_oil
    OilCooler.T_out=T2
    OilCooler.p_out=P1
    OilCooler.s_in=s1
    OilCooler.s_out=s2

    OilCooler.h_out=xL*h2_oil +(1.0-xL)*h2_ref
    OilCooler.mdot_total=mdot_ref+mdot_oil
class EvaporatorVals:
    def __init__(self):
        fields=['h_in','s_in','T_out','p_out','mdot',
            'T_L','Q','Edot','s_out','T_in']
        for f in fields:
            self.__dict__[f] = None
    def update(self,**kwargs):
        for f in kwargs:
            if f in self.__dict__:
```

```
                self.__dict__[f] = kwargs[f]
            else:
                print "Field "+f+" not found"
def Evaporator(Ref,Liq,Evaporator):
    #Retrieve struct values
    T2 = Evaporator.T_out
    P2 = Evaporator.p_out
    h1 = Evaporator.h_in
    s1 = Evaporator.s_in
    mdot = Evaporator.mdot
    T_L = Evaporator.T_L

    T0 = 298 #[K]
    h2 = Enthalpy_mix(Ref,Liq,T2,P2,0)
    s2 = Entropy_mix(Ref,Liq,T2,P2,0)
    Q = mdot * (h2-h1)
    Edot = T0 * (mdot*(s2-s1)-Q/T_L)
    #Store Struct Values
    Evaporator.Q=Q
    Evaporator.s_out=s2
    Evaporator.h_out=h2
    Evaporator.Edot=Edot
    Evaporator.p_in=P2
class OilExpansionVals:
    def __init__(self):
        fields=['T_in','T_evap','p_in','p_out','mdot_liq','mdot_ref'
            ,'eta_m']
        for f in fields:
            self.__dict__[f] = None
    def update(self,**kwargs):
        for f in kwargs:
            if f in self.__dict__:
                self.__dict__[f] = kwargs[f]
            else:
                print "Field "+f+" not found"
def OilExpansion(Ref,Liq,Exp):
    mdot_oil=Exp.mdot_liq #Not the total
    mdot_ref=Exp.mdot_ref
    T_in=Exp.T_in
    T_evap=Exp.T_evap
    p_in=Exp.p_in
    p_out=Exp.p_out
    eta_m=Exp.eta_m

    T0=298
    Wdot_ref=0
    Wdot_oil=0
    # Oil Part
    rho_in_oil = Density_mix(Ref,Liq,T_in,p_in,1.0)
    s_in_oil = Entropy_mix(Ref,Liq,T_in,p_in,1)
    h_in_oil = Enthalpy_mix(Ref,Liq,T_in,p_in,1)
    if mdot_oil>0:
        Wdot_oil = -mdot_oil/rho_in_oil*(p_out-p_in)*eta_m
        h_out_oil = Wdot_oil/mdot_oil+h_in_oil
        T_out_oil = T_hp(Ref,Liq,h_out_oil,p_out,1.0,T_in)
        s_out_oil = Entropy_mix(Ref,Liq,T_out_oil,p_out,1.0)
        Edot_oil=T0*mdot_oil*(s_out_oil-s_in_oil)
    else:
        Wdot=0.0
        Edot_oil=0.0
    # Refrigerant Part
    if mdot_ref>0:
        s_in_ref = Entropy_mix(Ref,Liq,T_in,p_in,0.0)
        h_in_ref = Enthalpy_mix(Ref,Liq,T_in,p_in,0.0)
        x_out_ref=Q_Ts(Ref,Liq,T_evap,s_in_ref,0.2)
        h_out_s_ref=Props('H','T',T_evap,'Q',x_out_ref,Ref)
        h_out_ref=h_in_ref+(h_out_s_ref-h_in_ref)*eta_m
```

```
        Wdot_ref=mdot_ref*(h_out_ref-h_in_ref)
        x_out_ref=Q_Th(Ref,Liq,T_evap,h_out_ref,0.2)
        s_out_ref=Props('S','T',T_evap,'Q',x_out_ref,Ref)
        Edot_ref=T0*mdot_ref*(s_out_ref-s_in_ref)
    else:
        Wdot_ref=0.0
        Edot_ref=0.0
    Exp.Edot=Edot_oil+Edot_ref
    Exp.Wdot=Wdot_ref+Wdot_oil
class RegeneratorVals:
    def __init__(self):
        fields=['Thi','phi','Tci','pci','epsilon','mdot']
        for f in fields:
            self.__dict__[f] = None
    def update(self,**kwargs):
        for f in kwargs:
            if f in self.__dict__:
                self.__dict__[f] = kwargs[f]
            else:
                print "Field "+f+" not found"
def Regenerator(Ref,Liq,Reg):
    # Assumes that Regenerator is all gas
    # Assumes that there are equal mass flow rates over both sides
       of regenerator
    T0=298 #%[K]
    Thi=Reg.Thi
    phi=Reg.phi
    Tci=Reg.Tci
    pci=Reg.pci
    epsilon=Reg.epsilon
    mdot=Reg.mdot
    hhi = Enthalpy_mix(Ref,Liq,Thi,phi,0)
    hci = Enthalpy_mix(Ref,Liq,Tci,pci,0)
    shi = Entropy_mix(Ref,Liq,Thi,phi,0)
    sci = Entropy_mix(Ref,Liq,Tci,pci,0)
    DELTAh_max_Reg = min(hhi-Enthalpy_mix(Ref,Liq,Tci,phi,0),
       Enthalpy_mix(Ref,Liq,Thi,pci,0)-hci)
    Q_reg=epsilon*mdot*DELTAh_max_Reg
    hho=hhi-epsilon*DELTAh_max_Reg
    hco=hci+epsilon*DELTAh_max_Reg
    Tho=T_hp(Ref,Liq,hho,phi,0,Tci)
    Tco=T_hp(Ref,Liq,hco,pci,0,Thi)
    sho = Entropy_mix(Ref,Liq,Tho,phi,0)
    sco = Entropy_mix(Ref,Liq,Tco,pci,0)
    Edot_Reg=T0*mdot*((sho+sco)-(shi+sci))
    Reg.Edot=Edot_Reg
    Reg.hho=hho
    Reg.hco=hco
    Reg.hhi=hhi
    Reg.hci=hci
    Reg.pco=pci
    Reg.pho=phi
    Reg.sci=sci
    Reg.sco=sco
    Reg.shi=shi
    Reg.sho=sho
    Reg.Tho=Tho
if __name__== "__main__":
    Ref='R744'
    Liq='PAO'
    p1=Props('P','T',273,'Q',1,Ref)
    (T2,h1,h2,s1,s2,xL2,Wdot,Edot)=Compressor(Ref,Liq,273,p1
       ,10000,0.5,0.7,0.13)
```

```
    print (T2,h1,h2,s1,s2,xL2,Wdot,Edot)
    (h2, T2, P2,Qh,Edot)=Condenser(Ref,Liq,300,500 ,290, 1,5,0.1)
    print (h2, T2, P2,Qh,Edot)

## ####################################################
##           FloodedCycleProps.py
## ####################################################
from CoolProp.CoolProp import Props, Tcrit
from math import log
from scipy import optimize
from numpy import float64
def Entropy_mix(Ref, Liq, T, P, xL):
    """
    Entropy of the mixture as a function of temperature [K]
    and pressure [kPa].  Output in kJ/kg-K
    """
    s_L = 0.
    T0 = 273.
    P0 = 101.325
    #Liquid Properties
    if Liq == "PAO":
        s_L = 1.940 * log(T/T0)
    elif Liq == "PAG":
        # PAG 0-OB-1020 from Tribology Data Handbook
        # T in K, cp in kJ/kg-K
        try:
            s_L=2.74374E-03*(T-T0)+1.08646*log(T/T0)
        except:
            a=4
    elif Liq == "POE":
        s_L = 2.30 * log(T/T0)
    elif Liq == "Water":
        cl_A=92.053
        cl_B=-0.039953
        cl_C=-0.00021103
        cl_D=5.3469E-07
        MM_l=18.0153
        try:
            s_L=(cl_A*log(T/298.15) + cl_B*(T-298.15) + cl_C/2.0*(T*
               T-298.15*298.15) + cl_D/3.0*(T*T*T
               -298.15*298.15*298.15))/MM_l
        except:
            a=1
    else:
        print "Invalid fluid"
    s_G = Props('S','T',float(T),'P',float(P),Ref)
    s = xL*s_L+(1-xL)*s_G
    return s
def Enthalpy_mix(Ref, Liq, T, P, xL):
    """
    Enthalpy of the mixture as a function of temperature [K]
    and pressure [kPa].  Output in kJ/kg
    """
    T0 = 273.0
    P0 = 101.325
    h = 0
    h_L = 0
    h_G = 0
    if Liq =='PAO':
        h_L = 1.940*(T-T0)+(P-P0)/849
    elif Liq=='PAG':
        # PAG 0-OB-1020 from Tribology Data Handbook
        rho_L=-0.726923*T+1200.22;
        h_L=2.74374E-03*(T**2-T0**2)/2.0+1.08646*(T-T0)+(P-P0)/rho_L
            ;
    elif Liq == 'POE':
        # From Totten, p 261, cp=0.55 cal/g-C --> 2.30 kJ/kg-K
        h_L = 2.30*(T-T0)+(P-P0)/930
```

```
    elif Liq =='Water':
        cl_A=92.053
        cl_B=-0.039953
        cl_C=-0.00021103
        cl_D=5.3469E-07
        MM_l=18.0153
        h_L=(cl_A*(T-298.15) + cl_B/2.0*(T**2-298.15**2) + cl_C
            /3.0*(T**3-298.15**3) + cl_D/4.0*(T**4-298.15**4))/MM_l+(
            P-P0)/Density_mix(Ref,Liq,T,P,1.0)
    else:
        print "Invalid fluid"
    h_G = Props('H','T',float(T),'P',float(P),Ref)
    if (xL==0):
        h = h_G
    elif (xL==1):
        h = h_L
    else:
        h = xL*h_L+(1-xL)*h_G
    return h
def Density_mix(Ref,Liq,T,P,xL,**kwargs):
    """
    Density of the mixture assuming homogeneous mixture properties
    """
    rho_L=0.0
    #Liquid Properties
    if Liq == 'PAO':
        rho_L=849
    elif Liq == 'PAG':
        # PAG O-OB-1020 from Tribology Data Handbook
        rho_L=-0.726923*T+1200.22;
    elif Liq == 'POE':
        rho_L=930
    elif Liq =="Water":
        # Water props from Yaws
        rhol_A=0.3471
        rhol_B=0.274
        rhol_n=0.28571
        rhol_Tc=647.13
        rho_L=rhol_A/pow(rhol_B,pow(1-T/rhol_Tc,rhol_n))*1000;
    else:
        print "Invalid fluid"
    rho_G=Props('D','T',float(T),'P',float(P),Ref)
    rho=None
    S=None
    alpha=None
    vL=1.0/rho_L
    vG=1.0/rho_G
    xG=1.0-xL
    x=xG
    if 'model' not in kwargs or kwargs['model']=='HEM':
        S=1
    elif kwargs['model']=='Zivi':
        S=(vG/vL)**(0.33333) #Eqn. 4.57 from Chisholm
    elif kwargs['model']=='Fauske':
        S=(vG/vL)**(0.5) #Eqn. 4.57 from Chisholm
    rho=(x+S*(1.0-x))/(x*vG+S*(1.0-x)*vL) #Eq 2.36 from Chisholm
    if x>0:
        alpha=1.0/(1.0+(1.0-x)/x*rho_G/rho_L*S)
    else:
        alpha=0.0
    if 'alphaOn' in kwargs and kwargs['alphaOn']==True:
        return rho,alpha
    else:
        return rho
def cp_mix(Ref,Liq,T,P,xL):
    cp_L = 0.0
    cp_G = 0.0
```

```
    xL = float(xL)
    if Liq ==  'PAO':
        cp_L = 1.940
    elif Liq == 'POE':
        cp_L = 2.30
    elif Liq == 'PAG':
        # PAG O-OB-1020 from Tribology Data Handbook
        # T in K, cp in kJ/kg-K
        cp_L=2.74374E-03*T+1.08646;
    elif Liq == "Water":
        MM_l=18.0153
        cl_A=92.053/MM_l
        cl_B=-0.039953/MM_l
        cl_C=-0.00021103/MM_l
        cl_D=5.3469E-07/MM_l
        cp_L= cl_A + cl_B*T + cl_C*T*T + cl_D*T*T*T
    cp_G=Props('C','T',float(T),'P',float(P),Ref)
    return xL*cp_L+(1.0-xL)*cp_G
def T_sp(Ref, Liq, s, p, xL, T_guess):
    """
    Solve for the temperature which gives the same entropy
    s [kJ/kg-K]
    p [kPa]
    T [K]
    """
    global Entropy_mix
    """
    When fsolve() is called, it will use a single element
    ndarray but we only want a double to be passed along,
    so take the first element of the 1-element array, which
    is a 64-bit float
    """
    f = lambda T: Entropy_mix(Ref, Liq, T[0], p, xL)-s
    T = optimize.fsolve(f,T_guess)
    return float(T)
def T_hp(Ref, Liq, h, p, xL, T_guess):
    """
    Solve for the temperature which gives the same enthalpy
    h [kJ/kg]
    p [kPa]
    T [K]
    """
    global Enthalpy_mix
    """
    When fsolve() is called, it will use a single element
    ndarray but we only want a double to be passed along,
    so take the first element of the 1-element array, which
    is a 64-bit float
    """
    f = lambda T: Enthalpy_mix(Ref, Liq, T[0], p, xL)-h
    T=optimize.fsolve(f,T_guess)
    return float(T)
def Q_Th(Ref,Liq, T, h, Q_guess):
    """
    Solve for the two-phase quality of refrigerant in the dome for a
        given
    enthalpy and saturation temperature
    Caveat: Function only good for pure refrigerant
    """
    """
    When fsolve() is called, it will use a single element
    ndarray but we only want a double to be passed along,
    so take the first element of the 1-element array, which
    is a 64-bit float
    """
    hl=Props('H','T',T,'Q',0,Ref)
```

```
    hv=Props('H','T',T,'Q',1,Ref)
    return (h-hl)/(hv-hl)
def Q_Ts(Ref,Liq, T, s, Q_guess):
    """
    Solve for the two-phase quality of refrigerant in the dome for a
        given
    enthalpy and saturation temperature
    Caveat: Function only good for pure refrigerant
    """
    """
    When fsolve() is called, it will use a single element
    ndarray but we only want a double to be passed along,
    so take the first element of the 1-element array, which
    is a 64-bit float
    """
    f = lambda Q : Props('S','T',T,'Q',Q[0],Ref)-s
    Q = optimize.fsolve(f,Q_guess)
    return float(Q)
def T_crit(Ref):
    """
    Critical Temperature of the refrigerant [K]
    """
    if Ref == 'R410A':
        T=Tcrit('R410A')
    elif Ref == 'R744':
        T=304.128
    elif Ref == 'R404A':
        T=Tcrit('R404A')
    elif Ref == 'R134a':
        T=374.2
    elif Ref == 'R502':
        T=355.3
    elif Ref == 'R290':
        T=369.82
    elif Ref == 'R717':
        T=405.4
    else:
        print 'Uh oh...  Refrigerant not found T_crit'
    return T
def Solubility_Ref_in_Liq(Ref,Liq,T,p):
    x_Ref=0.0
    Tmax=273.15
    Tmin=273.15
    error=False
    if Ref=='R744' and Liq=='Water':
        Tmin=273.15+40
        Tmax=273.15+100
        x_CO2=(8.47565584E-01-6.36371603E-03*T
        +1.84478093E-05*T**2-2.21498670E-08*T**3
        +8.76225186E-12*T**4-5.10876533E-05*p
        +3.57823191E-09*p**2-1.37995983E-13*p**3
        +2.05230067E-18*p**4+2.52713006E-07*T*p
        -7.95947422E-12*T*p**2+8.30553293E-17*T*p**3
        -6.90483618E-10*T**2*p+2.36780380E-14*T**2*p**2
        -2.84627591E-19*T**2*p**3+6.06021155E-13*T**3*p
        -2.13606178E-17*T**3*p**2+2.75290092E-22*T**3*p**3)
        x_Ref=x_CO2
    elif Ref=='R744' and Liq =='PAG':
        Tmin=273.15+40
        Tmax=273.15+100
        #kPa to bar
        p/=100
        x_40c_pag=+9.099439359-1.777194511588e+01*p**(0.96261)
            +15.8417564431*p-0.00785617429085*p**2+9.29355174098e-06*
            p**3
        x_100c_pag=+326.6783782-3.230924121929e+02*p**(0.0067777)
            +0.284830938229*p-0.00120728637563*p**2+2.90768469832e
            -06*p**3
        x_Ref=((x_100c_pag-x_40c_pag)/60*(T-313.15)+x_40c_pag)/100
```

```
    elif Ref=='R744' and Liq=='PAO':
        Tmin=273.15+40
        Tmax=273.15+100
        p/=100
        x_40c_pao=+0.08197431533-1.627640711357e+01*p**(0.99157)
            +16.0542422174*p-0.00239828193696*p**2+1.88834705177e-06*
            p**3
        x_100c_pao=+139.8515113-1.401455321692e+02*p**(0.001384)
            +0.117510290184*p-0.000176084086229*p**2+4.04345850972e
            -07*p**3
        x_Ref=((x_100c_pao-x_40c_pao)/60*(T-313.15)+x_40c_pao)/100
    elif Ref=='R744' and Liq=='POE':
        Tmin=273.15+40
        Tmax=273.15+100
        p/=100
        x_40c_poe=-0.07347897661-9.364025183662e+00*p**(0.99171)
            +9.34087009398*p+0.00230160520305*p**2-1.73258741724e-05*
            p**3
        x_100c_poe=-0.161283727+1.134483722323e+01*p**(0.99097)
            -10.9522884211*p+0.00316840572421*p**2-9.72225995465e-06*
            p**3
        x_Ref=((x_100c_poe-x_40c_poe)/60*(T-313.15)+x_40c_poe)/100
    else:
        print "Ref/Liquid [%s/%s] not implemented" %(Ref,Liq)
        x_Ref=0.0
    if T<Tmin or T>Tmax:
#        print "Error: T[%0.2f] out of range [%0.2f,%0.2f]" %(T,Tmin
    ,Tmax)
        error=True
    return x_Ref,error
if __name__== "__main__":
    print Entropy_mix("R744","PAG",300.,3000.,0.5)
    print Enthalpy_mix("R744","PAG",300.,3000.,0.5)
    print cp_mix("R744","PAG",300.,3000.,0)
    print Density_mix("R744","PAG",300.,3000.,0)
    h1 = Enthalpy_mix("R744","PAG",300,3000,0.5)
    s1 = Entropy_mix("R744","PAG",300,3000,0.5)
    T2s = T_sp("R744","PAG",s1,10000.0,0.5,300.0)
    print h1,s1,T2s
```

Appendix B: Appendices For Geometric Model

B.1 Fitting Of Curves

If manufacturer data is available for the curves which form the involutes, it is possible to use this data directly. In general it is more likely that the involute parameters (involute angles, base circle radius, etc.) will not be known *a priori*. In the case that the involute geometry must be determined from data, the first step is to obtain either accurate coordinate-measuring machine (CMM) data, or an optical scan of one of the scroll wraps. The CMM data is easier to post-process because the measurements are already in physical units rather than being in pixel coordinates, while it is faster and easier to obtain optical data. In this case, the scroll wrap is simply placed on a flatbed document scanner and scanned at high resolution. Figure

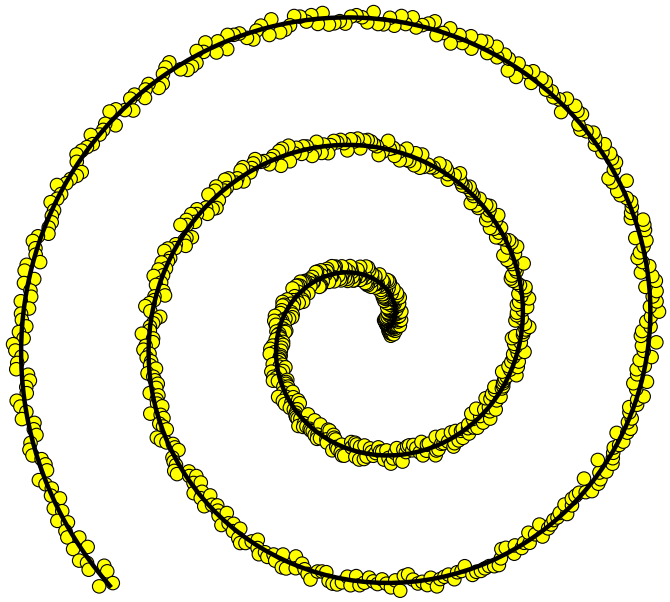


Figure B.1. Involute point cloud with exaggerated scatter.

B.1 shows an exaggerated point cloud representing one of the involutes of one of the scrolls. In general the scatter of the data is much less severe, even for optical scans. If an optical scan is used, a manual point-picking algorithm can be used to obtain the pixel coordinates of the points along the involute. It is suggested that if optical scans are used, the image is flipped and rotated so that the involute is swept in a clockwise fashion from the origin like that shown here. Orienting the scroll involutes with the analytical definition of an involute simplifies the fitting analysis.

To begin, it is assumed that the point cloud for the involute to be fit has been converted to a consistent set of physical length measurement units (e.g meters). The involute curve to be fit can be rotated, scaled, and translated in order to best fit the point cloud. The generic involute curve coordinates are given by the form

$$\begin{aligned} x &= r_b \left(\cos\phi + (\phi - \phi_0)\sin\phi\right) \\ y &= r_b \left(\sin\phi - (\phi - \phi_0)\cos\phi\right) \end{aligned} \tag{B.1}$$

The coordinates of the involute can be translated and rotated; introducing the translation and rotation coordinate transformations yields

$$\begin{aligned} x^* &= x\cos\zeta - y\sin\zeta + x_{tr} \\ y^* &= x\sin\zeta + y\cos\zeta + y_{tr} \end{aligned} \tag{B.2}$$

The major challenge of fitting the data is then to find a set of fit parameters (r_b, ϕ_0, ζ, x_{tr}, y_{tr}) that fit the point cloud data best. The "goodness of fit" is determined by calculating the root-mean-square error of the fit. The involute curve is discretized with ϕ taking on a wide enough range of values to ensure full coverage of the point cloud, and the distance of the point to the transformed involute curve is obtained by finding the point along the involute that is closest to the given point in the cloud. The root-mean-square error is obtained from

$$\varepsilon_{fit} = \sqrt{\frac{\sum\limits_{i=0..N-1} d_i^2}{N}} \tag{B.3}$$

where d_i is the distance of the i-th point in the cloud to the involute curve. An optimization routine is then used to minimize ε_{fit} by altering the parameters r_b, ϕ_0,

ζ, x_{tr}, and y_{tr}. It may be necessary to do some manual optimization to get rough values for the initial guess values after which the optimization routine can take over the rest of the computations. The Python code used to fit the involute curves is shown below

Figure B.2. Scanned Sanden Scroll orbiting scroll with overlaid curves.

Python code for involute fitting:

```
import matplotlib
from matplotlib import pyplot as plt
from matplotlib.figure import Figure
from matplotlib.axes import Subplot
import numpy as np
from scipy.optimize import fmin_l_bfgs_b,fmin,fmin_cg,anneal
class InvoluteOptimizer:
    def __init__(self,XDATA,YDATA):
        self.XDATA=XDATA
        self.YDATA=YDATA
        self.phi_i0=0.0
    def OBJECTIVE(self,x):
        x1=self.XDATA
        y1=self.YDATA
        rb=x[0]
        x0=x[1]
        y0=x[2]
        theta=0.0
```

```
        phi_i0=x[3]
        #If the scrolls are mirrored, (the involute curve is
            backwards), then flip the x coords
        mf=-1
        xx=x
        phi=np.linspace(self.phi_i0,25,10000)
        x=rb*cos(phi)+rb*(phi-phi_i0)*sin(phi)
        y=rb*sin(phi)-rb*(phi-phi_i0)*cos(phi)
        x2=cos(theta)*x-sin(theta)*y+x0
        y2=-sin(theta)*x-cos(theta)*y+y0
        """
        x1 is a row vector, so stack[tile] N copies of x1 vertically
        where N is the length of x2.  From that subtract, a column
            vector of
        x2 tiled to the right the length of x1 times. This yields a
            matrix of
        all of the differences in x and y directions between all
            points.
        """
        DX=np.tile(x1,(len(x2),1))-np.tile(x2.reshape(len(x2),1),(1,
            len(x1)))
        DY=np.tile(y1,(len(y2),1))-np.tile(y2.reshape(len(y2),1),(1,
            len(y1)))
        # Calculate the distance matrix between every point on the
            involute and
        # all the scroll points
        D=np.sqrt(np.power(DX,2)+np.power(DY,2))
        """
        For each point, find the minimal distance between the scroll
             point
        (scroll points are along the rows of the distance matrix
            with axis index of
        0) and all the points on the curve.  Then take the rms of
            the minimal
        distances which gives the rms of the fit
        """
        resid=np.sqrt(np.mean(np.power(D.min(0),2)))
        print resid,xx
        return resid
    def runSolver(self,wrap,**kwargs):
        if wrap=='outer':
            #For the outer involute:
            x0=[15.,1000.,1300.,0.3]
        else:
            x0=[15.,1000.,1300.,1.7]
        if 'solve' in kwargs:
            xf=fmin_cg(self.OBJECTIVE,np.array(x0),maxiter=20)
            return xf
        else:
            return x0
if __name__=='__main__':
    import pylab
    import numpy as np
    from numpy import cos, sin, pi
    INNER=np.loadtxt('inner_fixed.csv',delimiter=',')
    OUTER=np.loadtxt('outer_fixed.csv',delimiter=',')
    wraps=['outer','inner']
    XDATA=[]
    YDATA=[]
    coeffs=[]
    for wrap in wraps:
        if wrap=='outer':
            L=np.shape(OUTER)[1]
            XDATA=OUTER[0,:]
            YDATA=OUTER[1,:]
```

```
        IO=InvoluteOptimizer(XDATA,YDATA)
        xf=IO.runSolver('outer')
        coeffs.append(xf)
    else:
        XDATA=INNER[0,:]
        YDATA=INNER[1,:]
        IO=InvoluteOptimizer(XDATA,YDATA)
        xf=IO.runSolver('inner')
        coeffs.append(xf)
```

B.2 Angle Conversions

Morishita's model (1984), also used by Tseng (2006) assumes that ϕ_{o0}=-ϕ_{i0} , from which the parameters can be converted by

$$r_b = \frac{p_t}{2\pi} \tag{B.4}$$

$$t = 2r_b\phi_{i0} \tag{B.5}$$

$$\phi_{ie} = \phi_r = 2\pi\left(N + \frac{1}{4}\right) \tag{B.6}$$

using Morishita's definitions, the displacement of the compressor can be given by

$$V_{disp,mori} = (2N-1)\pi p_t(p_t - 2t)h \tag{B.7}$$

which yields the same displacement formula with substitution (the sum of the initial angles is zero).

B.3 Derivation Of Suction Break Angles

Derivation of ϕ_{s-sa} based on line from origin

To begin, the intersection angle is assumed to be of the form

$$\phi_{s,sa} = \phi_{ie} - \pi + B \tag{B.8}$$

The coordinates of the ending point of the fixed scroll are therefore equal to

$$x_{s,sa} = -r_b\left(\cos\phi_{s,sa} + (\phi_{s,sa} - \phi_{i0})\sin\phi_{s,sa}\right) + r_o\cos\left(\phi_{ie} - \pi/2 - \theta\right) \quad \text{(B.9)}$$

$$y_{s,sa} = -r_b\left(\sin\phi_{s,sa} - (\phi_{s,sa} - \phi_{i0})\cos\phi_{s,sa}\right) + r_o\sin\left(\phi_{ie} - \pi/2 - \theta\right) \quad \text{(B.10)}$$

and the coordinates of the point at the end of the inner involute of the fixed scroll are

$$x_e = r_b\left(\cos\phi_{ie} + (\phi_{ie} - \phi_{i0})\sin\phi_{ie}\right) \quad \text{(B.11)}$$

$$y_e = r_b\left(\sin\phi_{ie} - (\phi_{ie} - \phi_{i0})\cos\phi_{ie}\right) \quad \text{(B.12)}$$

If the vector from the origin to (x_e,y_e) is co-incident with the vector from the origin to ($x_{s,sa}$,$y_{s,sa}$), the vector cross-product will be equal to zero, or stated another way

$$0 = x_{s,sa}y_e - y_{s,sa}x_e \quad \text{(B.13)}$$

Substituting coordinates into Eqn. B.13 and dividing through by r_b^2, the result is

$$0 = \begin{pmatrix} -\left(\phi_{i0}\,B - \phi_e\,B - \phi_{i0}\,\phi_{o0} + \phi_e\,\phi_{o0} + \phi_e\,\phi_{i0} - \pi\,\phi_{i0} - \phi_e^2 + \pi\,\phi_e - 1\right)\,\sin\left(B\right) \\ -\left(B - \phi_{o0} + \phi_{i0} - \pi\right)\,\cos\left(B\right) + \dfrac{r_o}{r_b}\,\left(\phi_{i0}\sin\left(\theta\right) - \phi_e\sin\left(\theta\right) - \cos\left(\theta\right)\right) \end{pmatrix} \quad \text{(B.14)}$$

B is typically a small angle less than 0.15 radians, so it is acceptable to use the small-angle assumption ($\sin(B) \approx B$, $\cos B \approx 1$) which yields then

$$0 = \begin{pmatrix} \left(\phi_e - \phi_{i0}\right)\,B^2 + \left(\left(\phi_{i0} - \phi_e\right)\,\phi_{o0} + \left(\pi - \phi_e\right)\,\phi_{i0} + \phi_e^2 - \pi\,\phi_e\right)\,B \\ + \left(\phi_{i0} - \phi_e\right)\,\dfrac{r_o}{r_b}\sin\left(\theta\right) - \dfrac{r_o}{r_b}\cos\left(\theta\right) + \dfrac{r_o}{r_b} \end{pmatrix} \quad \text{(B.15)}$$

defining $b = -\phi_{o0} + \phi_e - \pi$ and

$$D = \frac{\left(\phi_{i0} - \phi_e\right)\,\dfrac{r_o}{r_b}\sin\left(\theta\right) - \dfrac{r_o}{r_b}\cos\left(\theta\right) + \dfrac{r_o}{r_b}}{\phi_e - \phi_{i0}} \quad \text{(B.16)}$$

Eqn. B.15 can be simplified to

$$0 = B^2 + bB + D \quad \text{(B.17)}$$

which has the solution from the quadratic equation of

$$B = \frac{-b + \sqrt{b^2 - 4D}}{2} \tag{B.18}$$

where the sign in front of the radical was selected to yield the proper solution. The derivative of B with respect to θ is needed for the volume derivative and is equal to

$$\frac{dB}{d\theta} = -\frac{r_o}{r_b} \frac{\sin(\theta) + (\phi_{i0} - \phi_e)\ \cos(\theta)}{(\phi_e - \phi_{i0})\ \sqrt{b^2 - 4\,D}} \tag{B.19}$$

Derivation of ϕ_{s-sa} based on tangent from involute

The location of the angle which divides the suction chamber and the suction area is critical to a definition of the suction chamber. This involute angle can be found by drawing a line from the base circle of the fixed scroll at a circle angle of ϕ_{ie} to the inner ending involute point on the fixed scroll, as shown in Figure B.3. The coordinates of the point on the base circle are equal to $(r_b \cos\phi_{ie}, r_b \sin\phi_{ie})$. Equating the slope of the tangent of the base circle and the slope of the line from the point on the base circle to the involute angle ϕ_{s-sa} yields

$$\frac{dy}{dx} = \underbrace{\frac{-\cos\phi_e}{\sin\phi_e}}_{\text{Circle}} = \underbrace{\frac{-\sin\phi_{s-sa} - \sin\phi_e + (\phi_{s-sa} - \phi_{i0})\cos\phi_{s-sa} - (\pi - \phi_{i0} + \phi_{o0})\cos(\phi_e - \theta)}{-\cos\phi_{s-sa} - \cos\phi_e - (\phi_{s-sa} - \phi_{i0})\sin\phi_{s-sa} + (\pi - \phi_{i0} + \phi_{o0})\sin(\phi_e - \theta)}}_{\text{Circle to } \phi_{s-sa}} \tag{B.20}$$

After simplification, and application of trigonometric substitutions, the equation to be solved is

$$\cos(\phi_{s-sa} - \phi_e) + (\phi_{s-sa} - \phi_{o0})\sin(\phi_{s-sa} - \phi_e) + (\pi - \phi_{i0} + \phi_{o0})\sin\theta = -1 \tag{B.21}$$

Equation B.21 does not have an analytic solution. If the solution for ϕ_{s-sa} is assumed to be a small perturbation around the angle $\phi_e - \pi$, then a preliminary solution for ϕ_{s-sa} can be expressed as

$$\phi_{s-sa} = B + \phi_e - \pi \tag{B.22}$$

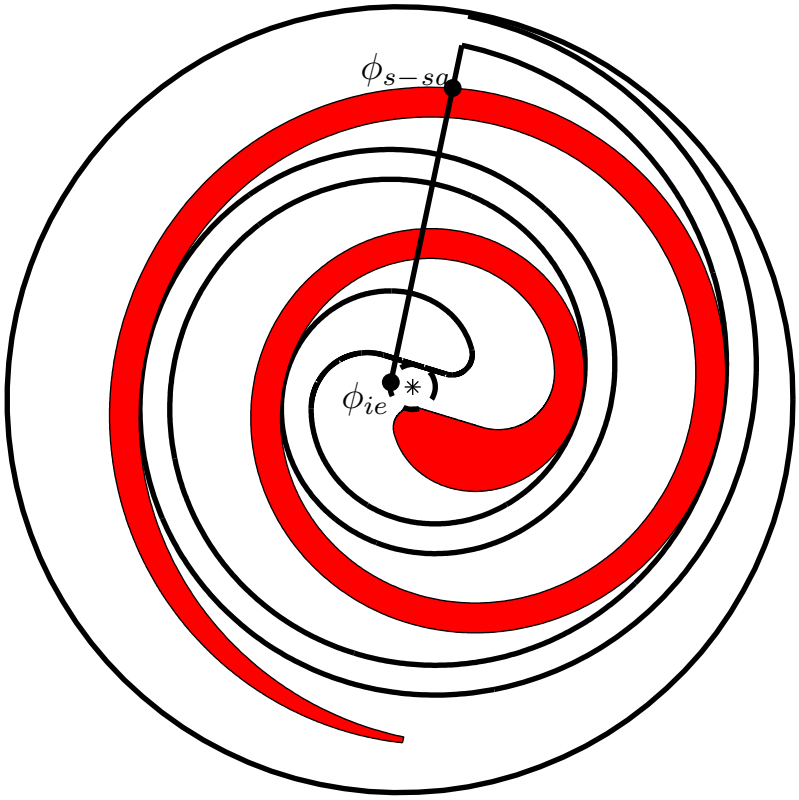


Figure B.3. Description of ϕ_{s-sa}.

Substitution of Equation B.22 into Equation B.21 yields

$$-\cos B - (B + \phi_e - \pi - \phi_{o0}) \sin B + (\pi - \phi_{i0} + \phi_{o0}) \sin \theta = -1 \tag{B.23}$$

From a consideration of the numerical solution it is seen that B is quite small (on the order of 0.15 radians), and it is fair to make the small-angle assumption as well as invoking the assumption that $B << \phi_e - \pi - \phi_{o0}$. Thus after solving for B and substituting, the solution for ϕ_{s-sa} is

$$\phi_{s-sa} = \frac{r_o/r_b}{\phi_e - \phi_{o0} - \pi} \sin \theta + \phi_e - \pi \tag{B.24}$$

In reality this is only an approximate solution because of the assumptions made, but in this case, the assumptions made are good. Figure B.4 shows that approximate and numerical solutions for ϕ_{s-sa} agree closely, with a maximum error of less than 0.01%.

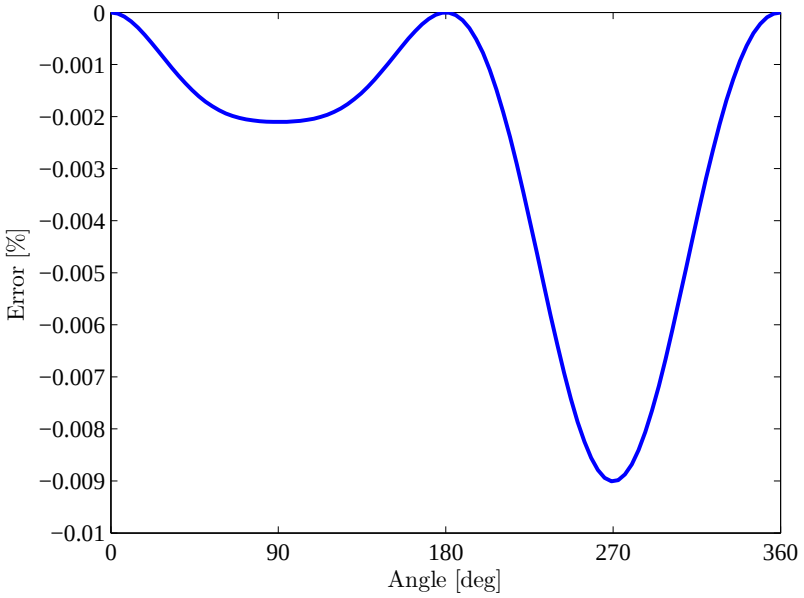


Figure B.4. Error between approximate and numerical solutions for ϕ_{s-sa}.

$$\frac{\mathbf{f}_{x,sa}}{p_{sa}} = r_b\, h_s\, \left(\sin\left(B+\phi_e\right) - \left(B-\phi_{o0}+\phi_e-\pi\right)\cos\left(B+\phi_e\right) + \cos\left(\phi_e\right)\,\phi_{o0} + \sin\left(\phi_e\right) - \phi_e\cos\left(\phi_e\right)\right) \tag{B.25}$$

$$\frac{\mathbf{f}_{y,sa}}{p_{sa}} = -r_b\, h_s\, \left(-\left(-B+\phi_{o0}-\phi_e+\pi\right)\sin\left(B+\phi_e\right) + \cos\left(B+\phi_e\right) - \sin\left(\phi_e\right)\,\phi_{o0} + \phi_e\sin\left(\phi_e\right) + \cos\left(\phi_e\right)\right) \tag{B.26}$$

$$\frac{\mathbf{f}_{x,s1}}{p_{s1}} = -r_b\, h_s\, \left(\sin\left(B+\phi_e\right) - \left(B-\phi_{o0}+\phi_e-\pi\right)\cos\left(B+\phi_e\right) + \sin\left(\theta-\phi_e\right) - \left(\theta+\phi_{o0}-\phi_e+\pi\right)\cos\left(\theta-\phi_e\right)\right) \tag{B.27}$$

$$\frac{\mathbf{f}_{y,s1}}{p_{s1}} = r_b\, h_s\, \left(\left(B-\phi_{o0}+\phi_e-\pi\right)\sin\left(B+\phi_e\right) + \cos\left(B+\phi_e\right) - \left(\theta+\phi_{o0}-\phi_e+\pi\right)\sin\left(\theta-\phi_e\right) - \cos\left(\theta-\phi_e\right)\right) \tag{B.28}$$

$$\frac{\mathbf{f}_{x,s2}}{p_{s2}} = -r_b\, h_s\, \left(\sin\left(\theta-\phi_e\right) - \left(\theta+\phi_{i0}-\phi_e\right)\cos\left(\theta-\phi_e\right) + \cos\left(\phi_e\right)\,\left(\phi_{i0}-\phi_e\right) + \sin\left(\phi_e\right)\right) \tag{B.29}$$

$$\frac{\mathbf{f}_{y,s2}}{p_{s2}} = -r_b\, h_s\, \left(\left(\theta+\phi_{i0}-\phi_e\right)\sin\left(\theta-\phi_e\right) + \cos\left(\theta-\phi_e\right) + \sin\left(\phi_e\right)\,\left(\phi_{i0}-\phi_e\right) - \cos\left(\phi_e\right)\right) \tag{B.30}$$

$$\frac{\mathbf{f}_{x,c2,\alpha}}{p_{c2,\alpha}} = \frac{\mathbf{f}_{x,c1,\alpha}}{p_{c1,\alpha}} = 2\,\pi\, r_b\, h_s \cos\left(\theta-\phi_e\right) \tag{B.31}$$

$$\frac{\mathbf{f}_{x,c2,\alpha}}{p_{c2,\alpha}} = \frac{\mathbf{f}_{y,c1,\alpha}}{p_{c1,\alpha}} = -2\,\pi\, r_b\, h_s \sin\left(\theta-\phi_e\right) \tag{B.32}$$

$$\frac{\mathbf{f}_{x,d1}}{p_{d1}} = r_b\, h_s\, \left(\sin\left(\theta - \phi_e\right) + \left(-\theta - \phi_{o0} + \phi_e - 2\,\pi\, N_c - \pi\right)\, \cos\left(\theta - \phi_e\right) - \sin\left(\phi_{os}\right) - \left(\phi_{o0} - \phi_{os}\right)\, \cos\left(\phi_{os}\right)\right) \tag{B.33}$$

$$\frac{\mathbf{f}_{y,d1}}{p_{d1}} = -r_b\, h_s\, \left(\left(-\theta - \phi_{o0} + \phi_e - 2\,\pi\, N_c - \pi\right)\, \sin\left(\theta - \phi_e\right) - \cos\left(\theta - \phi_e\right) - \left(\phi_{os} - \phi_{o0}\right)\, \sin\left(\phi_{os}\right) - \cos\left(\phi_{os}\right)\right) \tag{B.34}$$

$$\frac{\mathbf{f}_{x,d2}}{p_{d2}} = -r_b\, h_s\, \left(-\sin\left(\theta - \phi_e\right) + \left(\theta + \phi_{i0} - \phi_e + 2\,\pi\, N_c\right)\, \cos\left(\theta - \phi_e\right) + \sin\left(\phi_{os}\right) - \left(\phi_{os} - \phi_{i0} + \pi\right)\, \cos\left(\phi_{os}\right)\right) \tag{B.35}$$

$$\frac{\mathbf{f}_{y,d2}}{p_{d2}} = r_b\, h_s\, \left(\left(\theta + \phi_{i0} - \phi_e + 2\,\pi\, N_c\right)\, \sin\left(\theta - \phi_e\right) + \cos\left(\theta - \phi_e\right) - \left(-\phi_{os} + \phi_{i0} - \pi\right)\, \sin\left(\phi_{os}\right) + \cos\left(\phi_{os}\right)\right) \tag{B.36}$$

$$\frac{\mathbf{f}_{x,involute,dd}}{p_{dd}} = -r_b h_s\, \left(-\sin\left(\phi_{os}\right) + \left(\phi_{os} - \phi_{i0} + \pi\right)\, \cos\left(\phi_{os}\right) - \sin\left(\phi_{is}\right) - \left(\phi_{i0} - \phi_{is}\right)\, \cos\left(\phi_{is}\right)\right) \tag{B.37}$$

$$\frac{\mathbf{f}_{y,involute,dd}}{p_{dd}} = r_b h_s\, \left(\left(-\phi_{os} + \phi_{i0} - \pi\right)\, \sin\left(\phi_{os}\right) - \cos\left(\phi_{os}\right) - \left(\phi_{is} - \phi_{i0}\right)\, \sin\left(\phi_{is}\right) - \cos\left(\phi_{is}\right)\right) \tag{B.38}$$

B.5 Geometric Model Verification Data

In the tables that follow, a set of analytic data is used in order to allow for validation of the equations and the code. If there are any discrepancies between the equations in Chapter 4 and the code that follows the tables, the Python code from this section should be taken to be correct since it is properly validated against numerical results. The geometry employed is that of the Sanden compressor, whose geometry is defined in Section 4.13.

For the numerical results, the centroids and volumes of the chambers were obtained from high-resolution polygons that were used to form the outer edge of each chamber. For the force components, the product of the normal vector times the area of each element of the edge was used to form the numerical force component. The numerical volume derivative is obtained with a forward difference.

The difference between numeric and analytic solutions is in general less than 0.001%, the only exception being the suction chamber. In the suction chamber, a numerical solver is used to find the break angle, rather than using the approximate value.

Table B.1 Suction chamber s_1 Geometric Data.

	Analytic							(Analytic/Numeric-1)×100						
θ	V	$dV/d\theta$	c_x	c_y	f_x/p	f_y/p	M_O/p	V	$dV/d\theta$	c_x	c_y	f_x/p	f_y/p	M_O/p
rad	cm^3	cm^3/rad	m	m	kN/kPa	kN/kPa	kNm/kPa	%	%	%	%	%	%	%
0.0000	0.0000	-0.0000	N/A	N/A	-0.000000	0.000000	0.000E+00	N/A	-100.000	N/A	N/A	N/A	N/A	N/A
0.3927	0.1286	0.9884	0.0126	0.0516	-0.000250	-0.000630	2.408E-06	1.092	1.976	-0.647	-0.341	0.000	0.000	0.000
0.7854	0.9770	3.5989	0.0172	0.0490	-0.000675	-0.001103	4.703E-06	0.726	0.702	-0.512	-0.208	0.000	0.000	0.000
1.1781	3.0594	7.1376	0.0214	0.0454	-0.001204	-0.001354	6.850E-06	0.464	0.214	-0.356	-0.114	0.000	0.000	0.000
1.5708	6.5912	10.8278	0.0251	0.0410	-0.001752	-0.001359	8.828E-06	0.252	-0.029	-0.203	-0.047	0.000	0.000	0.000
1.9635	11.5016	14.0363	0.0281	0.0359	-0.002237	-0.001130	1.063E-05	0.105	-0.136	-0.087	-0.010	0.000	0.000	0.000
2.3562	17.5121	16.3599	0.0303	0.0304	-0.002590	-0.000715	1.227E-05	0.027	-0.147	-0.023	0.001	0.000	0.000	0.000
2.7489	24.2251	17.5966	0.0316	0.0246	-0.002763	-0.000186	1.377E-05	0.002	-0.094	-0.002	0.001	0.000	0.000	0.000
3.1416	31.1923	17.6737	0.0321	0.0187	-0.002742	0.000370	1.517E-05	0.000	-0.005	-0.000	0.000	0.000	0.000	0.000
3.5343	37.9543	16.5881	0.0316	0.0129	-0.002543	0.000869	1.652E-05	-0.004	0.087	0.003	-0.005	0.000	0.000	0.000
3.9270	44.0637	14.3841	0.0302	0.0073	-0.002208	0.001241	1.785E-05	-0.016	0.146	0.014	-0.055	0.000	0.000	0.000
4.3197	49.1047	11.1750	0.0280	0.0019	-0.001802	0.001438	1.920E-05	-0.030	0.127	0.025	-0.474	0.000	0.000	0.000
4.7124	52.7305	7.2072	0.0249	-0.0032	-0.001393	0.001444	2.060E-05	-0.032	-0.046	0.026	0.374	0.000	0.000	0.000
5.1051	54.7272	2.9361	0.0209	-0.0080	-0.001044	0.001272	2.205E-05	-0.022	-0.738	0.017	0.120	0.000	0.000	0.000
5.4978	55.1011	-0.9579	0.0160	-0.0123	-0.000801	0.000960	2.354E-05	-0.009	3.186	0.006	0.034	0.000	0.000	0.000
5.8905	54.1480	-3.6953	0.0101	-0.0157	-0.000690	0.000564	2.504E-05	-0.001	0.561	0.001	0.003	0.000	0.000	0.000
6.2832	52.4472	-4.6616	0.0035	-0.0176	-0.000712	0.000150	2.651E-05	0.000	-0.000	0.000	0.000	0.000	0.000	0.000

Table B.2 Compression Chamber c_1 Geometric Data (θ_d=4.275 rad).

	Analytic							(Analytic/Numeric-1)×100						
θ	V	$dV/d\theta$	c_x	c_y	f_x/p	f_y/p	M_O/p	V	$dV/d\theta$	c_x	c_y	f_x/p	f_y/p	M_O/p
rad	cm^3	cm^3/rad	m	m	kN/kPa	kN/kPa	kNm/kPa	%	%	%	%	%	%	%
0.0000	52.4472	-4.6616	0.0035	-0.0176	-0.000712	0.000150	2.651E-05	0.001	-0.000	0.001	0.000	0.000	0.000	0.000
0.3927	50.6166	-4.6616	-0.0031	-0.0170	-0.000600	0.000411	2.550E-05	0.001	-0.000	-0.001	0.000	0.000	0.000	0.000
0.7854	48.7860	-4.6616	-0.0088	-0.0141	-0.000397	0.000610	2.450E-05	0.001	-0.000	-0.000	0.000	0.000	0.000	0.000
1.1781	46.9554	-4.6616	-0.0128	-0.0096	-0.000134	0.000715	2.349E-05	0.001	-0.000	-0.000	0.000	0.000	0.000	0.000
1.5708	45.1248	-4.6616	-0.0148	-0.0041	0.000150	0.000712	2.248E-05	0.001	-0.000	0.000	0.001	0.000	0.000	0.000
1.9635	43.2941	-4.6616	-0.0147	0.0015	0.000411	0.000600	2.148E-05	0.001	0.000	0.000	-0.001	0.000	0.000	0.000
2.3562	41.4635	-4.6616	-0.0126	0.0064	0.000610	0.000397	2.047E-05	0.001	-0.000	0.000	-0.000	0.000	0.000	0.000
2.7489	39.6329	-4.6616	-0.0091	0.0101	0.000715	0.000134	1.946E-05	0.001	-0.000	0.000	-0.000	0.000	0.000	0.000
3.1416	37.8023	-4.6616	-0.0046	0.0121	0.000712	-0.000150	1.846E-05	0.001	-0.000	0.001	0.000	0.000	0.000	0.000
3.5343	35.9717	-4.6616	-0.0001	0.0124	0.000600	-0.000411	1.745E-05	0.001	-0.000	0.040	0.000	0.000	0.000	0.000
3.9270	34.1411	-4.6616	0.0041	0.0111	0.000397	-0.000610	1.644E-05	0.001	-0.000	-0.000	0.000	0.000	0.000	0.000
4.3197	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
4.7124	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
5.1051	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
5.4978	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
5.8905	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
6.2832	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

Table B.3 Discharge Chamber d_1 Geometric Data.

	Analytic							(Analytic/Numeric-1)×100						
θ	V	$dV/d\theta$	c_x	c_y	f_x/p	f_y/p	M_O/p	V	$dV/d\theta$	c_x	c_y	f_x/p	f_y/p	M_O/p
rad	cm^3	cm^3/rad	m	m	kN/kPa	kN/kPa	kNm/kPa	%	%	%	%	%	%	%
0.0000	20.4623	-7.8929	0.0081	-0.0077	-0.001030	0.000362	8.830E-06	0.000	-0.000	0.000	0.000	0.000	0.000	0.000
0.3927	17.2666	-8.3247	0.0063	-0.0101	-0.000906	0.000654	7.708E-06	0.000	-0.000	0.000	0.000	0.000	0.000	0.000
0.7854	13.9762	-8.3607	0.0042	-0.0121	-0.000692	0.000863	6.649E-06	0.000	0.000	0.000	0.000	0.000	0.000	0.000
1.1781	10.7587	-7.9511	0.0019	-0.0136	-0.000432	0.000968	5.652E-06	0.000	0.000	0.000	0.000	0.000	0.000	0.000
1.5708	7.7877	-7.1139	-0.0007	-0.0147	-0.000168	0.000966	4.719E-06	0.000	0.000	-0.001	0.000	0.000	0.000	0.000
1.9635	5.2167	-5.9323	-0.0034	-0.0152	0.000057	0.000870	3.849E-06	0.000	0.000	-0.000	0.000	-0.000	0.000	0.000
2.3562	3.1560	-4.5416	-0.0061	-0.0152	0.000217	0.000707	3.041E-06	0.000	0.000	-0.000	0.000	0.000	0.000	0.000
2.7489	1.6553	-3.1093	-0.0088	-0.0145	0.000296	0.000512	2.297E-06	0.000	0.000	0.000	0.000	0.000	0.000	0.000
3.1416	0.6965	-1.8092	-0.0112	-0.0133	0.000294	0.000320	1.615E-06	0.000	0.000	0.000	0.000	0.000	0.000	0.000
3.5343	0.1968	-0.7947	-0.0132	-0.0115	0.000226	0.000159	9.961E-07	0.000	0.000	0.000	0.000	0.000	0.000	0.000
3.9270	0.0204	-0.1759	-0.0147	-0.0093	0.000114	0.000049	4.403E-07	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4.3197	32.3104	-4.6644	0.0073	0.0085	0.000119	-0.000719	1.538E-05	0.001	-0.000	-0.000	0.000	0.000	0.000	0.000
4.7124	30.4418	-4.9189	0.0094	0.0052	-0.000286	-0.000714	1.395E-05	0.001	-0.000	0.000	0.000	0.000	0.000	0.000
5.1051	28.4018	-5.5179	0.0103	0.0017	-0.000643	-0.000562	1.257E-05	0.000	-0.000	0.000	0.001	0.000	0.000	0.000
5.4978	26.0805	-6.3258	0.0102	-0.0018	-0.000902	-0.000298	1.126E-05	0.000	-0.000	0.000	-0.001	0.000	0.000	0.000
5.8905	23.4280	-7.1754	0.0094	-0.0049	-0.001033	0.000028	1.001E-05	0.000	-0.000	0.000	-0.000	0.000	-0.001	0.000
6.2832	20.4623	-7.8929	0.0081	-0.0077	-0.001030	0.000362	8.830E-06	0.000	-0.000	0.000	0.000	0.000	0.000	0.000

Table B.4 Discharge Chamber *dd* Geometric Data for Sanden discharge geometry.

	Analytic							(Analytic/Numeric-1)×100						
θ	V	$dV/d\theta$	c_x	c_y	f_x/p	f_y/p	M_O/p	V	$dV/d\theta$	c_x	c_y	f_x/p	f_y/p	M_O/p
rad	cm^3	cm^3/rad	m	m	kN/kPa	kN/kPa	kNm/kPa	%	%	%	%	%	%	%
0.0000	4.0628	3.7529	0.0007	0.0031	0.000427	-0.000812	-2.699E-06	0.001	-0.000	0.000	-0.000	0.000	0.000	0.000
0.3927	5.8434	5.1992	0.0018	0.0026	0.000427	-0.000812	-2.699E-06	0.000	-0.000	-0.000	0.000	0.000	0.000	0.000
0.7854	8.0421	5.8540	0.0027	0.0017	0.000427	-0.000812	-2.699E-06	0.000	-0.000	-0.000	-0.000	0.000	0.000	0.000
1.1781	10.3239	5.6175	0.0031	0.0006	0.000427	-0.000812	-2.699E-06	0.000	0.000	0.000	0.000	0.000	0.000	0.000
1.5708	12.3415	4.5259	0.0031	-0.0007	0.000427	-0.000812	-2.699E-06	0.000	0.000	0.000	-0.000	0.000	0.000	0.000
1.9635	13.7878	2.7452	0.0026	-0.0018	0.000427	-0.000812	-2.699E-06	0.000	0.000	0.000	-0.000	0.000	0.000	0.000
2.3562	14.4426	0.5466	0.0017	-0.0027	0.000427	-0.000812	-2.699E-06	0.000	0.001	0.000	-0.000	0.000	0.000	0.000
2.7489	14.2061	-1.7353	0.0006	-0.0031	0.000427	-0.000812	-2.699E-06	0.000	-0.000	0.000	0.000	0.000	0.000	0.000
3.1416	13.1145	-3.7529	-0.0007	-0.0031	0.000427	-0.000812	-2.699E-06	0.000	-0.000	-0.000	0.000	0.000	0.000	0.000
3.5343	11.3338	-5.1992	-0.0018	-0.0026	0.000427	-0.000812	-2.699E-06	0.000	-0.000	-0.000	0.000	0.000	0.000	0.000
3.9270	9.1352	-5.8540	-0.0027	-0.0017	0.000427	-0.000812	-2.699E-06	0.000	-0.000	-0.000	0.000	0.000	0.000	0.000
4.3197	6.8533	-5.6175	-0.0031	-0.0006	0.000427	-0.000812	-2.699E-06	0.000	0.000	-0.000	-0.000	0.000	0.000	0.000
4.7124	4.8357	-4.5259	-0.0031	0.0007	0.000427	-0.000812	-2.699E-06	0.001	0.000	0.000	0.000	0.000	0.000	0.000
5.1051	3.3894	-2.7452	-0.0026	0.0018	0.000427	-0.000812	-2.699E-06	0.001	0.000	-0.000	0.000	0.000	0.000	0.000
5.4978	2.7346	-0.5466	-0.0017	0.0027	0.000427	-0.000812	-2.699E-06	0.001	0.001	0.000	0.000	0.000	0.000	0.000
5.8905	2.9711	1.7353	-0.0006	0.0031	0.000427	-0.000812	-2.699E-06	0.001	-0.000	-0.000	0.000	0.000	0.000	0.000
6.2832	4.0628	3.7529	0.0007	0.0031	0.000427	-0.000812	-2.699E-06	0.001	-0.000	-0.000	0.000	0.000	0.000	0.000

Table B.5 Discharge Chamber *dd* Geometric Data with two-arc solution with r_{a2}=2 mm.

	Analytic							(Analytic/Numeric-1)×100						
θ	V	$dV/d\theta$	c_x	c_y	f_x/p	f_y/p	M_O/p	V	$dV/d\theta$	c_x	c_y	f_x/p	f_y/p	M_O/p
rad	cm^3	cm^3/rad	m	m	kN/kPa	kN/kPa	kNm/kPa	%	%	%	%	%	%	%
0.0000	8.5056	3.7529	0.0007	0.0031	0.000427	-0.000812	-2.845E-06	0.002	-0.000	0.000	0.000	0.001	0.001	0.001
0.3927	10.2862	5.1992	0.0018	0.0026	0.000427	-0.000812	-2.845E-06	0.001	-0.000	0.000	0.000	0.001	0.001	0.001
0.7854	12.4849	5.8540	0.0027	0.0017	0.000427	-0.000812	-2.845E-06	0.001	-0.000	0.000	0.000	0.001	0.001	0.001
1.1781	14.7667	5.6175	0.0031	0.0006	0.000427	-0.000812	-2.845E-06	0.001	0.000	-0.000	-0.000	0.001	0.001	0.001
1.5708	16.7843	4.5259	0.0031	-0.0007	0.000427	-0.000812	-2.845E-06	0.001	0.000	0.000	0.000	0.001	0.001	0.001
1.9635	18.2306	2.7452	0.0026	-0.0018	0.000427	-0.000812	-2.845E-06	0.001	0.000	0.000	0.000	0.001	0.001	0.001
2.3562	18.8854	0.5466	0.0017	-0.0027	0.000427	-0.000812	-2.845E-06	0.001	0.001	-0.000	0.000	0.001	0.001	0.001
2.7489	18.6489	-1.7353	0.0006	-0.0031	0.000427	-0.000812	-2.845E-06	0.001	-0.000	0.000	-0.000	0.001	0.001	0.001
3.1416	17.5573	-3.7529	-0.0007	-0.0031	0.000427	-0.000812	-2.845E-06	0.001	-0.000	0.000	0.000	0.001	0.001	0.001
3.5343	15.7766	-5.1992	-0.0018	-0.0026	0.000427	-0.000812	-2.845E-06	0.001	-0.000	-0.000	-0.000	0.001	0.001	0.001
3.9270	13.5780	-5.8540	-0.0027	-0.0017	0.000427	-0.000812	-2.845E-06	0.001	-0.000	0.000	0.000	0.001	0.001	0.001
4.3197	11.2961	-5.6175	-0.0031	-0.0006	0.000427	-0.000812	-2.845E-06	0.001	0.000	-0.000	-0.000	0.001	0.001	0.001
4.7124	9.2785	-4.5259	-0.0031	0.0007	0.000427	-0.000812	-2.845E-06	0.001	0.000	-0.000	-0.000	0.001	0.001	0.001
5.1051	7.8322	-2.7452	-0.0026	0.0018	0.000427	-0.000812	-2.845E-06	0.002	0.000	0.000	-0.000	0.001	0.001	0.001
5.4978	7.1774	-0.5466	-0.0017	0.0027	0.000427	-0.000812	-2.845E-06	0.002	0.001	0.000	0.000	0.001	0.001	0.001
5.8905	7.4139	1.7353	-0.0006	0.0031	0.000427	-0.000812	-2.845E-06	0.002	-0.000	-0.000	0.000	0.001	0.001	0.001
6.2832	8.5056	3.7529	0.0007	0.0031	0.000427	-0.000812	-2.845E-06	0.002	-0.000	0.000	0.000	0.001	0.001	0.001

Table B.6 Discharge Chamber *dd* Geometric Data with single arc.

	Analytic							(Analytic/Numerical-1)×100						
θ	V	$dV/d\theta$	c_x	c_y	f_x/p	f_y/p	M_O/p	V	$dV/d\theta$	c_x	c_y	f_x/p	f_y/p	M_O/p
rad	cm^3	cm^3/rad	m	m	kN/kPa	kN/kPa	kNm/kPa	%	%	%	%	%	%	%
0.0000	12.1495	3.7529	0.0007	0.0031	0.000427	-0.000812	-2.845E-06	0.001	-0.000	0.000	-0.000	0.001	0.001	0.001
0.3927	13.9302	5.1992	0.0018	0.0026	0.000427	-0.000812	-2.845E-06	0.001	-0.000	0.000	0.000	0.001	0.001	0.001
0.7854	16.1288	5.8540	0.0027	0.0017	0.000427	-0.000812	-2.845E-06	0.001	-0.000	0.000	0.000	0.001	0.001	0.001
1.1781	18.4107	5.6175	0.0031	0.0006	0.000427	-0.000812	-2.845E-06	0.001	0.000	0.000	0.000	0.001	0.001	0.001
1.5708	20.4283	4.5259	0.0031	-0.0007	0.000427	-0.000812	-2.845E-06	0.001	0.000	-0.000	-0.000	0.001	0.001	0.001
1.9635	21.8746	2.7452	0.0026	-0.0018	0.000427	-0.000812	-2.845E-06	0.001	0.000	0.000	0.000	0.001	0.001	0.001
2.3562	22.5294	0.5466	0.0017	-0.0027	0.000427	-0.000812	-2.845E-06	0.001	0.001	0.000	-0.000	0.001	0.001	0.001
2.7489	22.2929	-1.7353	0.0006	-0.0031	0.000427	-0.000812	-2.845E-06	0.001	-0.000	0.000	-0.000	0.001	0.001	0.001
3.1416	21.2013	-3.7529	-0.0007	-0.0031	0.000427	-0.000812	-2.845E-06	0.001	-0.000	0.000	0.000	0.001	0.001	0.001
3.5343	19.4206	-5.1992	-0.0018	-0.0026	0.000427	-0.000812	-2.845E-06	0.001	-0.000	-0.000	-0.000	0.001	0.001	0.001
3.9270	17.2219	-5.8540	-0.0027	-0.0017	0.000427	-0.000812	-2.845E-06	0.001	-0.000	-0.000	-0.000	0.001	0.001	0.001
4.3197	14.9401	-5.6175	-0.0031	-0.0006	0.000427	-0.000812	-2.845E-06	0.001	0.000	-0.000	-0.000	0.001	0.001	0.001
4.7124	12.9225	-4.5259	-0.0031	0.0007	0.000427	-0.000812	-2.845E-06	0.001	0.000	-0.000	0.000	0.001	0.001	0.001
5.1051	11.4762	-2.7452	-0.0026	0.0018	0.000427	-0.000812	-2.845E-06	0.002	0.000	-0.000	0.000	0.001	0.001	0.001
5.4978	10.8214	-0.5466	-0.0017	0.0027	0.000427	-0.000812	-2.845E-06	0.002	0.001	0.000	0.000	0.001	0.001	0.001
5.8905	11.0579	1.7353	-0.0006	0.0031	0.000427	-0.000812	-2.845E-06	0.002	-0.000	-0.000	0.000	0.001	0.001	0.001
6.2832	12.1495	3.7529	0.0007	0.0031	0.000427	-0.000812	-2.845E-06	0.001	-0.000	0.000	0.000	0.001	0.001	0.001

B.6 Code For Geometric Model Validation

```
# scrollCalcs.py
# (c) Ian Bell 2010
#
# Condensed code for conducting the geometric calculations
# required for scroll compressor geometry (not including leakage)
# and validating these equations against high-accuracy polygons
# This code dynamically generates the tables of geometry code
# validation data in the appendix
from pylab import sqrt,pi,sin,cos
import plotScrolls as ps
import numpy as np
import pylab
def fxA(rb,phi,phi0):
    return rb**3/3.0*(4.0*((phi-phi0)**2-2.0)*sin(phi)+(phi0-phi)*((
        phi-phi0)**2-8.0)*cos(phi))
def fyA(rb,phi,phi0):
    return rb**3/3.0*((phi0-phi)*((phi-phi0)**2-8.0)*sin(phi)-4.0*((
        phi-phi0)**2-2.0)*cos(phi))
def S1(theta,poly=True,**kwargs):
    geo=kwargs.get('geo',ps.LoadGeo())
    h=geo.h
    rb=geo.rb
    phi_ie=geo.phi_ie
    phi_e=geo.phi_ie
    phi_o0=geo.phi_o0
    phi_i0=geo.phi_i0
    ro=rb*(pi-phi_i0+phi_o0)
    b=(-phi_o0+phi_e-pi)
    D=ro/rb*((phi_i0-phi_e)*sin(theta)-cos(theta)+1)/(phi_e-phi_i0)
    B=1.0/2.0*(sqrt(b**2-4.0*D)-b)
    B_prime=-ro/rb*(sin(theta)+(phi_i0-phi_ie)*cos(theta))/((phi_e-
        phi_i0)*sqrt(b**2-4*D))
    V0=h*rb**2/6.0*((phi_e-phi_i0)**3-(phi_e-theta-phi_i0)**3)
    dV0=h*rb**2/2.0*((phi_e-theta-phi_i0)**2)
    cx_0=h/V0*(fxA(rb,phi_ie,phi_i0)-fxA(rb,phi_ie-theta,phi_i0))
    cy_0=h/V0*(fyA(rb,phi_ie,phi_i0)-fyA(rb,phi_ie-theta,phi_i0))
    VIa=h*rb**2/6.0*((phi_e-pi+B-phi_o0)**3-(phi_e-pi-theta-phi_o0)
        **3)
    dVIa=h*rb**2/2.0*((phi_e-pi+B-phi_o0)**2*B_prime+(phi_e-pi-theta
        -phi_o0)**2)
    cx_Ia=h/VIa*(fxA(rb,phi_ie-pi+B,phi_o0)-fxA(rb,phi_ie-pi-theta,
        phi_o0))
    cy_Ia=h/VIa*(fyA(rb,phi_ie-pi+B,phi_o0)-fyA(rb,phi_ie-pi-theta,
        phi_o0))
    VIb=h*rb*ro/2.0*((B-phi_o0+phi_e-pi)*sin(B+theta)+cos(B+theta))
    dVIb=h*rb*ro*(B_prime+1)/2.0*((phi_e-pi+B-phi_o0)*cos(B+theta)-
        sin(B+theta))
    cx_Ib=1.0/3.0*(-rb*(B-phi_o0+phi_e-pi)*sin(B+phi_e)-rb*cos(B+
        phi_e)-ro*sin(theta-phi_e))
    cy_Ib=1.0/3.0*(-rb*sin(B+phi_e)+rb*(B-phi_o0+phi_e-pi)*cos(B+
        phi_e)-ro*cos(theta-phi_e))
    VIc=h*rb*ro/2
    dVIc=0
    cx_Ic=1.0/3.0*(rb*(-theta-phi_o0+phi_e-pi)*sin(theta-phi_e)-ro*
        sin(theta-phi_e)-rb*cos(theta-phi_e))
    cy_Ic=1.0/3.0*(rb*sin(theta-phi_e)+rb*(-theta-phi_o0+phi_e-pi)*
        cos(theta-phi_e)-ro*cos(theta-phi_e))
```

```
    cx_I=-(cx_Ia*VIa+cx_Ib*VIb-cx_Ic*VIc)/(VIa+VIb-VIc)+ro*cos(
        phi_ie-pi/2.0-theta)
    cy_I=-(cy_Ia*VIa+cy_Ib*VIb-cy_Ic*VIc)/(VIa+VIb-VIc)+ro*sin(
        phi_ie-pi/2.0-theta)
    Vs=V0-(VIa+VIb-VIc)
    dVs=dV0-(dVIa+dVIb-dVIc)
    cx=(cx_0*V0-cx_I*(VIa+VIb-VIc))/Vs
    cy=(cy_0*V0-cy_I*(VIa+VIb-VIc))/Vs
    fx_p=-rb*h*(sin(B+phi_e)-(B-phi_o0+phi_e-pi)*cos(B+phi_e)+sin(
        theta-phi_e)-(theta+phi_o0-phi_e+pi)*cos(theta-phi_e))
    fy_p=rb*h*((B-phi_o0+phi_e-pi)*sin(B+phi_e)+cos(B+phi_e)-(theta+
        phi_o0-phi_e+pi)*sin(theta-phi_e)-cos(theta-phi_e))
    M_0=(h*rb**2*(B-theta-2*phi_o0+2*phi_e-2*pi)*(B+theta))/2
    if poly==True:
        ################ Polygon calculations ##################
        phi=np.linspace(phi_ie-theta,phi_ie,2000)
        (xi,yi)=ps.coords_inv(phi, geo, theta, 'fi')
        phi=np.linspace(phi_ie-pi+B,phi_ie-pi-theta,2000)
        (xo,yo)=ps.coords_inv(phi, geo, theta, 'oo')
        V_poly=h*ps.polyarea(np.r_[xi,xo,xi[0]], np.r_[yi,yo,yi[0]])
        (cx_poly,cy_poly)=ps.polycentroid(np.r_[xi,xo,xi[0]], np.r_[
            yi,yo,yi[0]])
        ################ Numerical Force Calculations ###########
        phi=np.linspace(phi_ie-pi+B,phi_ie-pi-theta,2000)
        nx=np.zeros_like(phi)
        ny=np.zeros_like(phi)
        (nx,ny)=ps.coords_norm(phi,geo,theta,'oo')
        L=len(xo)
        dA=h*np.sqrt(np.power(xo[1:L]-xo[0:L-1],2)+np.power(yo[1:L]-
            yo[0:L-1],2))
        dfxp_poly=dA*(nx[1:L]+nx[0:L-1])/2.0
        dfyp_poly=dA*(ny[1:L]+ny[0:L-1])/2.0
        fxp_poly=np.sum(dfxp_poly)
        fyp_poly=np.sum(dfyp_poly)
        rOx=xo-geo.ro*cos(phi_e-pi/2-theta)
        rOx=(rOx[1:L]+rOx[0:L-1])/2
        rOy=yo-geo.ro*sin(phi_e-pi/2-theta)
        rOy=(rOy[1:L]+rOy[0:L-1])/2
        MO_poly=np.sum(rOx*dfyp_poly-rOy*dfxp_poly)
    else:
        (V_poly,cx_poly,cy_poly,fxp_poly,fyp_poly,MO_poly)=(None,
            None,None,None,None,None)
    return Vs,dVs,cx,cy,fx_p,fy_p,M_0,(cx_Ia,cy_Ia,cx_Ib,cy_Ib,cx_Ic
        ,cy_Ic,cx_I,cy_I,cx_0,cy_0,V0,VIa,VIb,VIc),V_poly,cx_poly,
        cy_poly,fxp_poly,fyp_poly,MO_poly
def S2(theta,**kwargs):
    geo=kwargs.get('geo',ps.LoadGeo())
    h=geo.h
    rb=geo.rb
    phi_ie=geo.phi_ie
    phi_e=geo.phi_ie
    phi_o0=geo.phi_o0
    phi_i0=geo.phi_i0
    ro=rb*(pi-phi_i0+phi_o0)
    (Vs1,dVs1,cx_s1,cy_s1,fx_ps1,fy_ps1,M_0_s1,cs1,V_polys1,
        cx_polys1,cy_polys1,fxp_polys1,fyp_polys1,M_0s1_poly)=S1(
        theta)
    (cx,cy)=(-cx_s1+ro*cos(phi_ie-pi/2-theta),-cy_s1+ro*sin(phi_ie-
        pi/2-theta))
    b=(-phi_o0+phi_e-pi)
    D=ro/rb*((phi_i0-phi_e)*sin(theta)-cos(theta)+1)/(phi_e-phi_i0)
```

```
    B=1.0/2.0*(sqrt(b**2-4.0*D)-b)
    fx_p=-rb*h*(sin(theta-phi_e)-(theta+phi_i0-phi_e)*cos(theta-
        phi_e)+cos(phi_e)*(phi_i0-phi_e)+sin(phi_e))
    fy_p=-rb*h*((theta+phi_i0-phi_e)*sin(theta-phi_e)+cos(theta-
        phi_e)+sin(phi_e)*(phi_i0-phi_e)-cos(phi_e))
    M_O=(h*rb**2*theta*(theta+2*phi_i0-2*phi_e))/2
    ################ Numerical Force Calculations ############
    phi=np.linspace(phi_ie-theta,phi_ie,2000)
    (xo,yo)=ps.coords_inv(phi, geo, theta, 'oi')
    nx=np.zeros_like(phi)
    ny=np.zeros_like(phi)
    (nx,ny)=ps.coords_norm(phi,geo,theta,'oi')
    L=len(xo)
    dA=h*np.sqrt(np.power(xo[1:L]-xo[0:L-1],2)+np.power(yo[1:L]-yo
        [0:L-1],2))
    fxp_poly=np.sum(dA*(nx[1:L]+nx[0:L-1])/2.0)
    fyp_poly=np.sum(dA*(ny[1:L]+ny[0:L-1])/2.0)
    return (Vs1,dVs1,cx,cy,fx_p,fy_p,fxp_poly,fyp_poly)
def C1(theta,alpha,poly=True,**kwargs):
    geo=kwargs.get('geo',ps.LoadGeo())
    h=geo.h
    rb=geo.rb
    phi_ie=geo.phi_ie
    phi_e=geo.phi_ie
    phi_o0=geo.phi_o0
    phi_i0=geo.phi_i0
    ro=rb*(pi-phi_i0+phi_o0)
    ###################### Analytic Calculations ###################
    V=-pi*h*rb*ro*(2*theta+4*alpha*pi-2*phi_ie-pi+phi_i0+phi_o0)
    dV=-2.0*pi*h*rb*ro
    psi=rb/3.0*(3.0*theta**2+6.0*phi_o0*theta+3.0*phi_o0**2+pi
        **2-15.0+(theta+phi_o0)*(12.0*pi*alpha-6.0*phi_ie)+3.0*phi_ie
        **2+12.0*pi*alpha*(pi*alpha-phi_ie))/(2.0*theta+phi_o0-2.0*
        phi_ie+phi_i0+4.0*pi*alpha-pi)
    cx=-2.0*rb*cos(theta-phi_ie)-psi*sin(theta-phi_ie)
    cy=+2.0*rb*sin(theta-phi_ie)-psi*cos(theta-phi_ie)
    fx_p= 2.0*pi*rb*h*cos(theta-phi_e)
    fy_p=-2.0*pi*rb*h*sin(theta-phi_e)
    M_O=-2*pi*h*rb*rb*(theta+phi_o0-phi_e+2*pi*alpha)
    if poly==True:
        ###################### Polygon Calculations
            ####################
        phi=np.linspace(geo.phi_ie-theta-2*pi*alpha,geo.phi_ie-theta
            -2*pi*(alpha-1), 1000)
        (xi,yi)=ps.coords_inv(phi, geo, theta, 'fi')
        phi=np.linspace( geo.phi_ie-theta-2*pi*(alpha-1)-pi,geo.
            phi_ie-theta-2*pi*alpha-pi,1000)
        (xo,yo)=ps.coords_inv(phi, geo, theta, 'oo')
        V_poly=h*ps.polyarea(np.r_[xi,xo], np.r_[yi,yo])
        (cx_poly,cy_poly)=ps.polycentroid(np.r_[xi,xo], np.r_[yi,yo
            ])
        ###################### Force Calculations
            ##########################
        phi=np.linspace( geo.phi_ie-theta-2*pi*(alpha)-pi,geo.phi_ie
            -theta-2*pi*(alpha-1)-pi,1000)
        (xo,yo)=ps.coords_inv(phi, geo, theta, 'oo')
        nx=np.zeros_like(phi)
        ny=np.zeros_like(phi)
        (nx,ny)=ps.coords_norm(phi,geo,theta,'oo')
        L=len(xo)
        dA=h*np.sqrt(np.power(xo[1:L]-xo[0:L-1],2)+np.power(yo[1:L]-
            yo[0:L-1],2))
```

```
        dfxp_poly=dA*(nx[1:L]+nx[0:L-1])/2.0
        dfyp_poly=dA*(ny[1:L]+ny[0:L-1])/2.0
        fxp_poly=np.sum(dfxp_poly)
        fyp_poly=np.sum(dfyp_poly)
        rOx=xo-geo.ro*cos(phi_e-pi/2-theta)
        rOx=(rOx[1:L]+rOx[0:L-1])/2
        rOy=yo-geo.ro*sin(phi_e-pi/2-theta)
        rOy=(rOy[1:L]+rOy[0:L-1])/2
        MO_poly=np.sum(rOx*dfyp_poly-rOy*dfxp_poly)
    else:
         (V_poly,cx_poly,cy_poly,fxp_poly,fyp_poly,MO_poly)=(None,
            None,None,None,None,None)
    return (V,dV,cx,cy,fx_p,fy_p,M_O,V_poly,cx_poly,cy_poly,fxp_poly
        ,fyp_poly,MO_poly)
def C2(theta,alpha,**kwargs):
    geo=kwargs.get('geo',ps.LoadGeo())
    h=geo.h
    rb=geo.rb
    phi_ie=geo.phi_ie
    phi_e=geo.phi_ie
    phi_o0=geo.phi_o0
    phi_i0=geo.phi_i0
    ro=rb*(pi-phi_i0+phi_o0)
    geo=ps.LoadGeo()
    ro=geo.ro
    phi_ie=geo.phi_ie
    (Vc1,dVc1,cxc1,cyc1,fx_pc1,fy_pc1,M_Oc1,V_polyc1,cx_polyc1,
        cy_polyc1,fxp_polyc1,fyp_polyc1,M_Oc1_poly)=C1(theta,alpha)
    (cx,cy)=(-cxc1+ro*cos(phi_ie-pi/2-theta),-cyc1+ro*sin(phi_ie-pi
        /2-theta))
    fx_p= 2.0*pi*rb*h*cos(theta-phi_e)
    fy_p=-2.0*pi*rb*h*sin(theta-phi_e)
    M_O=2*pi*h*rb*rb*(theta+phi_i0-phi_e+2*pi*alpha-pi)
    ###################### Force Calculations
        ##########################
    phi=np.linspace( geo.phi_ie-theta-2*pi*(alpha),geo.phi_ie-theta
        -2*pi*(alpha-1),1000)
    (xo,yo)=ps.coords_inv(phi, geo, theta, 'oi')
    nx=np.zeros_like(phi)
    ny=np.zeros_like(phi)
    (nx,ny)=ps.coords_norm(phi,geo,theta,'oi')
    L=len(xo)
    dA=h*np.sqrt(np.power(xo[1:L]-xo[0:L-1],2)+np.power(yo[1:L]-yo
        [0:L-1],2))
    fxp_poly=np.sum(dA*(nx[1:L]+nx[0:L-1])/2.0)
    fyp_poly=np.sum(dA*(ny[1:L]+ny[0:L-1])/2.0)
    return (Vc1,dVc1,cx,cy,fx_p,fy_p,fxp_poly,fyp_poly)
def D1(theta,poly=True,**kwargs):
    geo=kwargs.get('geo',ps.LoadGeo())
    hs=geo.h
    rb=geo.rb
    phi_ie=geo.phi_ie
    phi_e=geo.phi_ie
    phi_o0=geo.phi_o0
    phi_i0=geo.phi_i0
    phi_is=geo.phi_is
    phi_os=geo.phi_os
    ro=rb*(pi-phi_i0+phi_o0)
    Nc=ps.Nc(theta,geo=geo)
    phi2=phi_ie-theta-2.0*pi*Nc
    phi1=phi_os+pi
    VO=hs*rb**2/6.0*((phi2-phi_i0)**3-(phi1-phi_i0)**3)
```

```
dVO=-hs*rb**2/2.0*((phi2-phi_i0)**2)
cx_O=hs/VO*(fxA(rb,phi2,phi_i0)-fxA(rb,phi1,phi_i0))
cy_O=hs/VO*(fyA(rb,phi2,phi_i0)-fyA(rb,phi1,phi_i0))

phi2=phi_ie-theta-2.0*pi*Nc-pi
phi1=phi_os
VIa=hs*rb**2/6.0*((phi2-phi_o0)**3-(phi1-phi_o0)**3)
dVIa=-hs*rb**2/2.0*((phi2-phi_o0)**2)
cx_Ia=hs/VIa*(fxA(rb,phi2,phi_o0)-fxA(rb,phi1,phi_o0))
cy_Ia=hs/VIa*(fyA(rb,phi2,phi_o0)-fyA(rb,phi1,phi_o0))

VIb=hs*rb*ro/2.0*((phi_os-phi_o0)*sin(theta+phi_os-phi_ie)+cos(
    theta+phi_os-phi_ie))
dVIb=hs*rb*ro/2.0*((phi_os-phi_o0)*cos(theta+phi_os-phi_ie)-sin(
    theta+phi_os-phi_ie))
cx_Ib=1.0/3.0*(-ro*sin(theta-phi_ie)+rb*(phi_os-phi_o0)*sin(
    phi_os)+rb*cos(phi_os))
cy_Ib=1.0/3.0*(-ro*cos(theta-phi_ie)-rb*(phi_os-phi_o0)*cos(
    phi_os)+rb*sin(phi_os))

VIc=hs*rb*ro/2.0
dVIc=0
cx_Ic=1.0/3.0*((rb*(-theta+phi_ie-phi_o0-2*pi*Nc-pi)-ro)*sin(
    theta-phi_ie)-rb*cos(theta-phi_ie))
cy_Ic=1.0/3.0*((rb*(-theta+phi_ie-phi_o0-2*pi*Nc-pi)-ro)*cos(
    theta-phi_ie)+rb*sin(theta-phi_ie))

VId= hs*rb*ro/2.0*((phi_os-phi_i0+pi)*sin(theta+phi_os-phi_ie)+
    cos(theta+phi_os-phi_ie)+1)
dVId=hs*rb*ro/2.0*((phi_os-phi_i0+pi)*cos(theta+phi_os-phi_ie)-
    sin(theta+phi_os-phi_ie))
cx_Id=(rb*(2*phi_os-phi_o0-phi_i0+pi)*sin(phi_os)-2*(ro*sin(
    theta-phi_ie)-rb*cos(phi_os)))/3.0
cy_Id=(-2*(ro*cos(theta-phi_ie)-rb*sin(phi_os))-rb*(2*phi_os-
    phi_o0-phi_i0+pi)*cos(phi_os))/3.0

VI=VIa+VIb+VIc+VId
dVI=dVIa+dVIb+dVIc+dVId
cx_I=-(cx_Ia*VIa+cx_Ib*VIb+cx_Ic*VIc+cx_Id*VId)/VI+ro*cos(phi_ie
    -pi/2.0-theta)
cy_I=-(cy_Ia*VIa+cy_Ib*VIb+cy_Ic*VIc+cy_Id*VId)/VI+ro*sin(phi_ie
    -pi/2.0-theta)

Vd1=VO-VI
dVd1=dVO-dVI
cx=(cx_O*VO-cx_I*VI)/Vd1
cy=(cy_O*VO-cy_I*VI)/Vd1

fx_p=rb*hs*(sin(theta-phi_e)+(-theta-phi_o0+phi_e-2*pi*Nc-pi)*
    cos(theta-phi_e)-sin(phi_os)-(phi_o0-phi_os)*cos(phi_os))
fy_p=-rb*hs*((-theta-phi_o0+phi_e-2*pi*Nc-pi)*sin(theta-phi_e)-
    cos(theta-phi_e)-(phi_os-phi_o0)*sin(phi_os)-cos(phi_os))
M_O=(hs*rb**2*(theta-phi_os+2*phi_o0-phi_e+2*pi*Nc+pi)*(theta+
    phi_os-phi_e+2*pi*Nc+pi))/2.0

if poly==True:
    ########################## Polygon calculations
        ##################
    phi=np.linspace(phi_os+pi,phi_ie-theta-2.0*pi*Nc,1000)
    (xi,yi)=ps.coords_inv(phi, geo, theta, 'fi')
    phi=np.linspace(phi_ie-theta-2.0*pi*Nc-pi,phi_os,1000)
    (xo,yo)=ps.coords_inv(phi, geo, theta, 'oo')
    V_poly=hs*ps.polyarea(np.r_[xi,xo], np.r_[yi,yo])
    (cx_poly,cy_poly)=ps.polycentroid(np.r_[xi,xo,xi[0]], np.r_[
        yi,yo,yi[0]])
    ###################### Force Calculations
        ##########################
    phi=np.linspace(phi_os,phi_ie-theta-2.0*pi*Nc-pi,1000)
    (xo,yo)=ps.coords_inv(phi, geo, theta, 'oo')
```

```
            nx=np.zeros_like(phi)
            ny=np.zeros_like(phi)
            (nx,ny)=ps.coords_norm(phi,geo,theta,'oo')
            L=len(xo)
            dA=hs*np.sqrt(np.power(xo[1:L]-xo[0:L-1],2)+np.power(yo[1:L
                ]-yo[0:L-1],2))
            dfxp_poly=dA*(nx[1:L]+nx[0:L-1])/2.0
            dfyp_poly=dA*(ny[1:L]+ny[0:L-1])/2.0
            fxp_poly=np.sum(dfxp_poly)
            fyp_poly=np.sum(dfyp_poly)
            rOx=xo-geo.ro*cos(phi_e-pi/2-theta)
            rOx=(rOx[1:L]+rOx[0:L-1])/2
            rOy=yo-geo.ro*sin(phi_e-pi/2-theta)
            rOy=(rOy[1:L]+rOy[0:L-1])/2
            MO_poly=np.sum(rOx*dfyp_poly-rOy*dfxp_poly)
        else:
            (V_poly,cx_poly,cy_poly,fxp_poly,fyp_poly,MO_poly)=(None,
                None,None,None,None,None)
        return Vd1,dVd1,cx,cy,fx_p,fy_p,M_O,(cx_Ia,cy_Ia,cx_Ib,cy_Ib,
            cx_Ic,cy_Ic,cx_Id,cy_Id,cx_I,cy_I,cx_O,cy_O,VO,VIa,VIb,VIc,
            VId,dVO,dVIa,dVIb,dVIc,dVId),V_poly,cx_poly,cy_poly,fxp_poly,
            fyp_poly,MO_poly
    def D2(theta,**kwargs):
        geo=kwargs.get('geo',ps.LoadGeo())
        h=geo.h
        rb=geo.rb
        phi_ie=geo.phi_ie
        phi_e=geo.phi_ie
        phi_o0=geo.phi_o0
        phi_i0=geo.phi_i0
        phi_is=geo.phi_is
        phi_os=geo.phi_os
        ro=rb*(pi-phi_i0+phi_o0)
        Nc=ps.Nc(theta,geo=geo)
        (Vd1,dVd1,cxd1,cyd1,fx_pd1,fy_pd1,M_Od1,cd1,V_polyd1,cx_polyd1,
            cy_polyd1,fxp_polyd1,fyp_polyd1,M_Od1_poly)=D1(theta)
        fx_p=-h*rb*(-sin(theta-phi_e)+(theta+phi_i0-phi_e+2*pi*Nc)*cos(
            theta-phi_e)+sin(phi_os)-(phi_os-phi_i0+pi)*cos(phi_os))
        fy_p=h*rb*((theta+phi_i0-phi_e+2*pi*Nc)*sin(theta-phi_e)+cos(
            theta-phi_e)-(-phi_os+phi_i0-pi)*sin(phi_os)+cos(phi_os))
        M_O=-(h*rb**2*(theta-phi_os+2*phi_i0-phi_e+2*pi*Nc-pi)*(theta+
            phi_os-phi_e+2*pi*Nc+pi))/2
        (cx,cy)=(-cxd1+ro*cos(phi_ie-pi/2-theta),-cyd1+ro*sin(phi_ie-pi
            /2-theta))
        phi=np.linspace(phi_os+pi,phi_ie-theta-2.0*pi*Nc,1000)
        (xo,yo)=ps.coords_inv(phi, geo, theta, 'oi')
        nx=np.zeros_like(phi)
        ny=np.zeros_like(phi)
        (nx,ny)=ps.coords_norm(phi,geo,theta,'oi')
        L=len(xo)
        dA=h*np.sqrt(np.power(xo[1:L]-xo[0:L-1],2)+np.power(yo[1:L]-yo
            [0:L-1],2))
        fxp_poly=np.sum(dA*(nx[1:L]+nx[0:L-1])/2.0)
        fyp_poly=np.sum(dA*(ny[1:L]+ny[0:L-1])/2.0)
        return Vd1,dVd1,cx,cy,fx_p,fy_p,fxp_poly,fyp_poly
    def DD(theta,Type,r2,poly=True,**kwargs):
        geo=kwargs.get('geo',ps.LoadGeo())
        ps.setDiscGeo(geo, Type, r2)
        hs=geo.h
        xa1=geo.xa_arc1
        ya1=geo.ya_arc1
```

```
ra1=geo.ra_arc1
ta1_2=geo.t2_arc1
ta1_1=geo.t1_arc1
xa2=geo.xa_arc2
ya2=geo.ya_arc2
ra2=geo.ra_arc2
ta2_2=geo.t2_arc2
ta2_1=geo.t1_arc2
ro=geo.ro
m_line=geo.m_line
b_line=geo.b_line
t1_line=geo.t1_line
t2_line=geo.t2_line
phi_os=geo.phi_os
phi_o0=geo.phi_o0
phi_i0=geo.phi_i0
phi_is=geo.phi_is
phi_e=geo.phi_ie
rb=geo.rb
om=geo.phi_ie-pi/2-theta
(xoos,yoos)=ps.coords_inv(geo.phi_os, geo, theta, 'oo')
#################### Oa portion ####################
V_Oa=hs*((-(ra1*(cos(ta1_2)*(ya1-yoos)-sin(ta1_2)*(xa1-xoos)-ra1
    *ta1_2))/2)-(-(ra1*(cos(ta1_1)*(ya1-yoos)-sin(ta1_1)*(xa1-
    xoos)-ra1*ta1_1))/2))
dV_Oa=-hs*ra1*ro/2.0*((sin(om)*sin(ta1_2)+cos(om)*cos(ta1_2))-(
    sin(om)*sin(ta1_1)+cos(om)*cos(ta1_1)))
#################### Ob portion ####################
x1l=t1_line #old nomenclature
y1l=m_line*t1_line+b_line #old nomenclature
V_Ob=hs/2.0*((ro*xoos-ro*x1l)*sin(om)-(ro*cos(om)-2.0*x1l)*yoos+
    y1l*(ro*cos(om)-2.0*xoos))
dV_Ob=ro*hs/2.0*(ro-yoos*sin(om)-xoos*cos(om)-y1l*sin(om)-x1l*
    cos(om))
#################### Oc portion ####################
V_Oc=rb*hs/6*(
      3*ro*(phi_os-phi_i0+pi)*sin(theta+phi_os-phi_e)
      +3*ro*cos(theta+phi_os-phi_e)
      +3*(phi_is-phi_i0)*ro*sin(theta+phi_is-phi_e)
      +3*ro*cos(theta+phi_is-phi_e)
      +3*rb*((phi_is-phi_i0)*(phi_os-phi_o0)+1)*sin(phi_os-
          phi_is)
      -3*rb*(phi_os-phi_o0-phi_is+phi_i0)*cos(phi_os-phi_is)
      +rb*((phi_os+pi-phi_i0)**3-(phi_is-phi_i0)**3)+3*ro)
dV_Oc=rb*hs*ro/2*(
       (phi_os-phi_i0+pi)*cos(theta+phi_os-phi_e)
      -sin(theta+phi_os-phi_e)
      +(phi_is-phi_i0)*cos(theta+phi_is-phi_e)
      -sin(theta+phi_is-phi_e)
      )
#################### Ia portion ####################
V_Ia=hs*ra2/2.0*(xa2*(sin(ta2_2)-sin(ta2_1))
             -ya2*(cos(ta2_2)-cos(ta2_1))
             -rb*(sin(ta2_2-phi_os)-sin(ta2_1-phi_os))
    -rb*(phi_os-phi_o0)*(cos(ta2_2-phi_os)-cos(ta2_1-phi_os))
             +ra2*(ta2_2-ta2_1)  )
dV_Ia=0.0
#################### Ib portion ####################
x1l=t1_line #old nomenclature
x2l=t2_line #old nomenclature
y1l=m_line*t1_line+b_line #old nomenclature
ml=m_line
```

```
V_Ib=-hs*(x2l-x1l)/2.0*(rb*ml*(cos(phi_os)+(phi_os-phi_o0)*sin(
    phi_os))+b_line-rb*(sin(phi_os)-(phi_os-phi_o0)*cos(phi_os)))
dV_Ib=0
cx=ro*cos(om)/2.0 #By symmetry
cy=ro*sin(om)/2.0 #By symmetry
V=2.0*(V_Oa+V_Ob+V_Oc-V_Ia-V_Ib)
dV=2.0*(dV_Oa+dV_Ob+dV_Oc-dV_Ia-dV_Ib)
################### Force Components #########
#Arc 1
fx_p =-hs*geo.ra_arc1*(sin(geo.t2_arc1)-sin(geo.t1_arc1))
fy_p =+hs*geo.ra_arc1*(cos(geo.t2_arc1)-cos(geo.t1_arc1))
M_O  =-hs*geo.ra_arc1*((sin(geo.t2_arc1)-sin(geo.t1_arc1))*geo.
    ya_arc1+(cos(geo.t2_arc1)-cos(geo.t1_arc1))*geo.xa_arc1)
#Arc 2
fx_p+=+hs*geo.ra_arc2*(sin(geo.t2_arc2)-sin(geo.t1_arc2))
fy_p+=-hs*geo.ra_arc2*(cos(geo.t2_arc2)-cos(geo.t1_arc2))
M_O +=+hs*geo.ra_arc2*((sin(geo.t2_arc2)-sin(geo.t1_arc2))*geo.
    ya_arc2+(cos(geo.t2_arc2)-cos(geo.t1_arc2))*geo.xa_arc2)
#Line
x1t=-geo.xa_arc1-geo.ra_arc1*cos(geo.t1_arc1)+ro*cos(om)
y1t=-geo.ya_arc1-geo.ra_arc1*sin(geo.t1_arc1)+ro*sin(om)
x2t=-geo.xa_arc2-geo.ra_arc2*cos(geo.t1_arc2)+ro*cos(om)
y2t=-geo.ya_arc2-geo.ra_arc2*sin(geo.t1_arc2)+ro*sin(om)
L=np.sqrt((x2t-x1t)**2+(y2t-y1t)**2)
if L>1e-12:
    Lx=(x2t-x1t)/L
    Ly=(y2t-y1t)/L
    nx=-1/np.sqrt(1+Lx**2/Ly**2)
    ny=Lx/Ly/np.sqrt(1+Lx**2/Ly**2)
    # Make sure you get the cross product with the normal
    # pointing towards the scroll, otherwise flip...
    if Lx*ny-Ly*nx<0:
        nx*=-1
        ny*=-1
    fx_p+=hs*nx*L
    fy_p+=hs*ny*L
    rx=(x1t+x2t)/2-ro*cos(om)
    ry=(y2t+y2t)/2-ro*sin(om)
    M_O+=rx*hs*ny*L-ry*hs*nx*L
#Involute portion
fx_p+=-hs*(-sin(phi_os)+(phi_os-phi_i0+pi)*cos(phi_os)-sin(
    phi_is)-(phi_i0-phi_is)*cos(phi_is))*rb
fy_p+=hs*((-phi_os+phi_i0-pi)*sin(phi_os)-cos(phi_os)-(phi_is-
    phi_i0)*sin(phi_is)-cos(phi_is))*rb
M_O +=-(hs*(phi_os-phi_is+pi)*(phi_os+phi_is-2*phi_i0+pi)*rb*rb)
    /2
if poly==True:
    ##############################################################
    ##                        POLYGON                           ##
    ##############################################################
    t=np.linspace(geo.t1_arc1,geo.t2_arc1,300)
    (x_farc1,y_farc1)=(
        geo.xa_arc1+geo.ra_arc1*cos(t),
        geo.ya_arc1+geo.ra_arc1*sin(t))
    (x_oarc1,y_oarc1)=(
       -geo.xa_arc1-geo.ra_arc1*cos(t)+geo.ro*cos(om),
       -geo.ya_arc1-geo.ra_arc1*sin(t)+geo.ro*sin(om))
    (nx_oarc1,ny_oarc1)=(-cos(t),-sin(t))
    t=np.linspace(geo.t1_arc2,geo.t2_arc2,300)
    (x_farc2,y_farc2)=(
        geo.xa_arc2+geo.ra_arc2*cos(t),
        geo.ya_arc2+geo.ra_arc2*sin(t))
    (x_oarc2,y_oarc2)=(
```

```
    -geo.xa_arc2-geo.ra_arc2*cos(t)+geo.ro*cos(om),
    -geo.ya_arc2-geo.ra_arc2*sin(t)+geo.ro*sin(om))
(nx_oarc2,ny_oarc2)=(+cos(t),+sin(t))
phi=np.linspace(phi_is,phi_os+pi,300)
(x_finv,y_finv)=ps.coords_inv(phi,geo,theta,'fi')
(x_oinv,y_oinv)=ps.coords_inv(phi,geo,theta,'oi')
(nx_oinv,ny_oinv)=ps.coords_norm(phi,geo,theta,'oi')
x=np.r_[x_farc2[::-1],x_farc1,x_finv,x_oarc2[::-1],x_oarc1,
    x_oinv,x_farc2[-1]]
y=np.r_[y_farc2[::-1],y_farc1,y_finv,y_oarc2[::-1],y_oarc1,
    y_oinv,y_farc2[-1]]
(cx_poly,cy_poly)=ps.polycentroid(x,y)
V_poly=geo.h*ps.polyarea(x, y)
fxp_poly=0
fyp_poly=0
MO_poly=0
#Arc1
L=len(nx_oarc1)
dA=hs*np.sqrt(np.power(x_oarc1[1:L]-x_oarc1[0:L-1],2)+np.
    power(y_oarc1[1:L]-y_oarc1[0:L-1],2))
dfxp_poly=dA*(nx_oarc1[1:L]+nx_oarc1[0:L-1])/2.0
dfyp_poly=dA*(ny_oarc1[1:L]+ny_oarc1[0:L-1])/2.0
fxp_poly=np.sum(dfxp_poly)
fyp_poly=np.sum(dfyp_poly)
rOx=x_oarc1-geo.ro*cos(phi_e-pi/2-theta)
rOx=(rOx[1:L]+rOx[0:L-1])/2
rOy=y_oarc1-geo.ro*sin(phi_e-pi/2-theta)
rOy=(rOy[1:L]+rOy[0:L-1])/2
MO_poly=np.sum(rOx*dfyp_poly-rOy*dfxp_poly)
#Arc2
L=len(nx_oarc2)
dA=hs*np.sqrt(np.power(x_oarc2[1:L]-x_oarc2[0:L-1],2)+np.
    power(y_oarc2[1:L]-y_oarc2[0:L-1],2))
dfxp_poly=dA*(nx_oarc2[1:L]+nx_oarc2[0:L-1])/2.0
dfyp_poly=dA*(ny_oarc2[1:L]+ny_oarc2[0:L-1])/2.0
fxp_poly+=np.sum(dfxp_poly)
fyp_poly+=np.sum(dfyp_poly)
rOx=x_oarc2-geo.ro*cos(phi_e-pi/2-theta)
rOx=(rOx[1:L]+rOx[0:L-1])/2
rOy=y_oarc2-geo.ro*sin(phi_e-pi/2-theta)
rOy=(rOy[1:L]+rOy[0:L-1])/2
MO_poly+=np.sum(rOx*dfyp_poly-rOy*dfxp_poly)
#Involute
L=len(y_oinv)
dA=hs*np.sqrt(np.power(x_oinv[1:L]-x_oinv[0:L-1],2)+np.power
    (y_oinv[1:L]-y_oinv[0:L-1],2))
dfxp_poly=dA*(nx_oinv[1:L]+nx_oinv[0:L-1])/2.0
dfyp_poly=dA*(ny_oinv[1:L]+ny_oinv[0:L-1])/2.0
fxp_poly+=np.sum(dfxp_poly)
fyp_poly+=np.sum(dfyp_poly)
rOx=x_oinv-geo.ro*cos(phi_e-pi/2-theta)
rOx=(rOx[1:L]+rOx[0:L-1])/2
rOy=y_oinv-geo.ro*sin(phi_e-pi/2-theta)
rOy=(rOy[1:L]+rOy[0:L-1])/2
MO_poly+=np.sum(rOx*dfyp_poly-rOy*dfxp_poly)
#Line
x1t=-geo.xa_arc1-geo.ra_arc1*cos(geo.t1_arc1)+ro*cos(om)
y1t=-geo.ya_arc1-geo.ra_arc1*sin(geo.t1_arc1)+ro*sin(om)
x2t=-geo.xa_arc2-geo.ra_arc2*cos(geo.t1_arc2)+ro*cos(om)
y2t=-geo.ya_arc2-geo.ra_arc2*sin(geo.t1_arc2)+ro*sin(om)
L=np.sqrt((x2t-x1t)**2+(y2t-y1t)**2)
if L>1e-12:
    Lx=(x2t-x1t)/L
```

```
            Ly=(y2t-y1t)/L
            nx=-1/np.sqrt(1+Lx**2/Ly**2)
            ny=Lx/Ly/np.sqrt(1+Lx**2/Ly**2)
            # Make sure you get the cross product with the normal
            # pointing towards the scroll, otherwise flip...
            if Lx*ny-Ly*nx<0:
                nx*=-1
                ny*=-1
            fxp_poly+=hs*nx*L
            fyp_poly+=hs*ny*L
            rx=(x1t+x2t)/2-ro*cos(om)
            ry=(y2t+y2t)/2-ro*sin(om)
            MO_poly+=rx*hs*ny*L-ry*hs*nx*L
    else:
        (V_poly,cx_poly,cy_poly,fxp_poly,fyp_poly,MO_poly)=(None,
            None,None,None,None,None)
    return V,dV,cx,cy,fx_p,fy_p,M_O,(V_Oa,dV_Oa,V_Ob,dV_Ob),V_poly,
        cx_poly,cy_poly,fxp_poly,fyp_poly,MO_poly
if __name__=='__main__':
    #Create new empty structure
    geo=ps.geoVals()
##      #Load up values
##      geo.rb=0.0018872
##      geo.h=0.02635
##      geo.phi_i0=1.8048
##      geo.phi_is=5.341
##      geo.phi_ie=20.61
##      geo.phi_o0=0.351
##      geo.phi_os=2.1994
##      geo.phi_oe=20.61
    #Set discharge geometry
    ps.setDiscGeo(geo,'2Arc',0.001)
    th=pi/2
    # Do all the calcs...
    (V,dV,cx,cy,fx_p,fy_p,M_O,c,V_poly,cx_poly,cy_poly,fxp_poly,
        fyp_poly,M_O_poly)=S1(th,geo=geo)
    print "s1"
    print fx_p,fxp_poly,fy_p,fyp_poly
    (V,dV,cx,cy,fx_p,fy_p,fxp_poly,fyp_poly)=S2(th,geo=geo)
    print "s2"
    print fx_p,fxp_poly,fy_p,fyp_poly
    (V,dV,cx,cy,fx_p,fy_p,M_O,V_poly,cx_poly,cy_poly,fxp_poly,
        fyp_poly,M_O_poly)=C1(th,1,geo=geo)
    print "c1,1"
    print fx_p,fxp_poly,fy_p,fyp_poly
    (V,dV,cx,cy,fx_p,fy_p,fxp_poly,fyp_poly)=C2(th,1,geo=geo)
    print "c2,1"
    print fx_p,fxp_poly,fy_p,fyp_poly
    (V,dV,cx,cy,fx_p,fy_p,c,V_poly,M_O,cx_poly,cy_poly,fxp_poly,
        fyp_poly,M_O_poly)=D1(th,geo=geo)
    print "d1"
    print fx_p,fxp_poly,fy_p,fyp_poly
    (V,dV,cx,cy,fx_p,fy_p,fxp_poly,fyp_poly)=D2(th,geo=geo)
    print "d2"
    print fx_p,fxp_poly,fy_p,fyp_poly
    (V,dV,cx,cy,fx_p,fy_p,M_O,c,V_poly,cx_poly,cy_poly,fxp_poly,
        fyp_poly,M_O_poly)=DD(th,'2Arc',0.000,geo=geo)
    print "dd"
    print fx_p,fxp_poly,fy_p,fyp_poly,M_O,M_O_poly
```

Appendix C: Appendices For Overall Model

C.1 Nitrogen Properties

Thermodynamic Properties

$$c_p = \frac{1}{MM}\left(\psi_1 + \psi_2 T + \psi_2 T^2 + \psi_3 T^3 + \psi_4 T^4\right)$$

$$c_v = \frac{1}{MM}\left(\psi_1 + \psi_2 T + \psi_2 T^2 + \psi_3 T^3 + \psi_4 T^4\right) - R$$

$$h = \frac{1}{MM}\left(\psi_1 T + \frac{\psi_2}{2}T^2 + \frac{\psi_3}{3}T^3 + \frac{\psi_4}{4}T^4 + \frac{\psi_5}{5}T^5 - 8671.61943\right)$$

$$u = h - RT$$

$$s = \frac{1}{MM}\left(\psi_1 \ln\left(\frac{T}{T_o}\right) + \psi_2 T + \frac{\psi_3}{2}T^2 + \frac{\psi_4}{3}T^3 + \frac{\psi_5}{4}T^4 + 0.6450354\right) - R\ln\left(\frac{P}{P_o}\right)$$

Table C.1 Constants for thermodynamic properties.

ψ_1	ψ_2	ψ_3	ψ_4	ψ_5	MM	R	T_o
29.342	-0.0035395	0.000010076	-4.3116E-09	2.5935E-13	28.01	0.297	298.15

Transport Properties

$$\mu = (42.606 + 0.475T - 0.0000988T^2)/1e7$$

with μ [Pa-s], T [K]

$$k = (0.00309 + 7.593\text{e-}5T - 1.1014\text{e-}8T^2)/1000$$

with k [kW/m-K], T [K]

C.2 Refrigerant Properties

Table C.2 Refrigerant property correlations employed.

Refrigerant	Equation of State	Viscosity	Thermal Conductivity
R134a	Tillner-Roth (1994) [a]	Scalabrin (2006b)	Scalabrin (2006a)
R290	Miyamoto (2000)	Scalabrin (2006c)	Marsh (2002)
R404A	Lemmon (2003)	Geller (2000)	Geller (2001)
R410A	Lemmon (2003)	Geller (2000)	Geller (2001)
R407C	Lemmon (2003)	Geller (2000)	Geller (2001)
R507A	Lemmon (2003)	Geller (2000)	Geller (2001)
R717	Tillner-Roth (1993)	Fenghour (1995)	Tufeu (1984)
R744	Span (1996)	Fenghour (1998)[b]	None
Nitrogen	Span (2000)	Lemmon (2004)	Lemmon (2004)
Argon	Tegeler (1999)	Lemmon (2004)	Lemmon (2004)

[a]Note: Equation 30 in Tillner-Roth et al. for second derivative of the residual Helmholtz function with respect to δ has a typo; term in parentheses in equation should be $2d_i + k - 1 - k\delta^k$

[b]Critical enhancement not implemented

C.3 On-the-fly Lookup Table Generation And Interpolation

One of the challenges of using the equation of state to provide thermodynamic properties is that the evaluation of the EOS for a given set of inputs is quite slow. This is particularly the case if temperature and pressure are the input state variables. All of the EOS listed in Table C.2 are formulated that properties are given as function of temperature and density, or expressed another way: $h = f(T, \rho)$. Therefore, if temperature and pressure are known, and it is desired to obtain the enthalpy, it is necessary to first solve for the density, then use the density and temperature to evaluate the enthalpy. Numerically, solving for the density for a known pressure and

temperature is computationally expensive, and makes the EOS difficult to implement into a more detailed model. For instance, in the detailed compressor model, the enthalpy function alone is called many millions of times, making the efficiency of the property code of utmost importance.

One elegant way of getting around this two-step process is to build lookup tables of the enthalpy, internal energy, density, etc. as a function of temperature and pressure, then do a two-dimensional interpolation process in order to find the properties at a given temperature and pressure. This process is only applied in the subcooled, superheated and supercritical regions of the temperature-pressure plane as these are the only locations where the thermodynamic properties are uniquely defined for a set of temperature and pressure.

A simplified example can help to explain the procedure. A 2-D matrix of temperature-pressure points is developed, and at each point, all relevant thermophysical properties are calculated, here taken to be the enthalpy. These are the ◯ points in Figure C.1. One the challenges with using interpolation in general is that the function may not be smooth enough. This becomes a problem when you have solvers nested inside solvers nested inside solvers like Russian nesting dolls. Often the outer solvers require derivatives of inner solver function, if only indirectly, which means that if the first and second derivatives of the surface for the inner solver are not smooth, it can tend to drive derivatives in the outer solver to very large values, potentially causing the solver to fail.

In this simple example, we are looking to calculate the enthalpy at a temperature of 337.5 K and a pressure of 375 kPa in a matrix of calculated enthalpies. First three

one-dimensional interpolations are carried out along the lines of constant pressure using the method

$$L_0 = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} \tag{C.1}$$

$$L_1 = \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} \tag{C.2}$$

$$L_2 = \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} \tag{C.3}$$

$$f^* = L_0 \cdot f_0 + L_1 \cdot f_1 + L_2 \cdot f_2 \tag{C.4}$$

where the x_i values are input values, x is the target input value, f_i are the output function values and f^* is the interpolated value of the output function at the target input value. Once the values of the enthalpy at a temperature of 337.5 K have been obtained through interpolation for all the pressures (the $\triangle$ points from Figure C.1), the 337.5 K isotherm is interpolated along to find the desired enthalpy at a pressure of 375 kPa (the $\star$ point from Figure C.1). This method is extremely efficient and fast, and allows for the integration of extremely high-accuracy and extremely slow equations of state into very detailed component models.

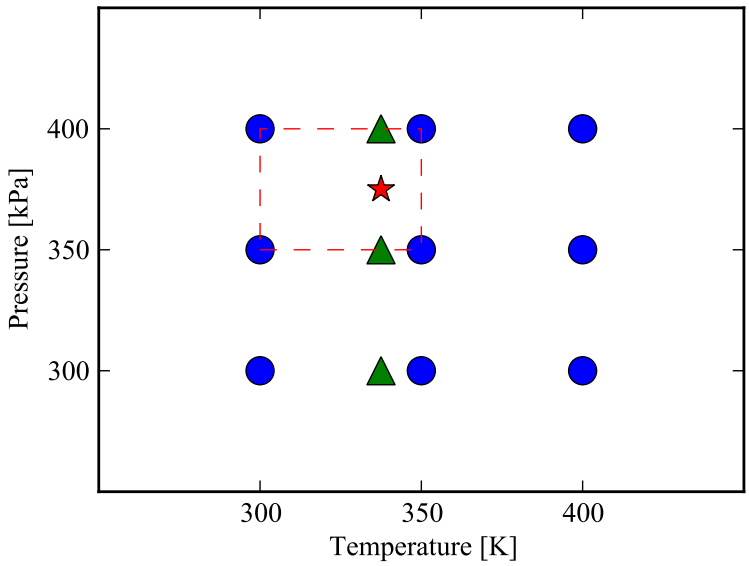


Figure C.1. Schematic of 2-D interpolation scheme (◯: Correlation points, △: First set of 1-D interpolations, ⋆: Solution point from second set of 1-D interpolation).

C.4 Liquid Properties

Thermodynamic Properties

The specific heat of the liquid can be given by the polynomial term

$$c = \sum_{i=0}^{3} c_i T^i \tag{C.5}$$

and the internal energy by

$$u = \int_{T=T_o}^{T} c(T)dT = \sum_{i=0}^{3} \frac{c_i}{i+1}(T^{i+1} - T_o^{i+1}) \tag{C.6}$$

and the enthalpy by

$$h = u + (p - p_0)/\rho \tag{C.7}$$

and the entropy by

$$s = \int_{T=T_o}^{T} \frac{c(T)}{T}dT = c_0 \ln\frac{T}{T_0} + \sum_{i=1}^{3} \frac{c_i}{i}(T^i - T_o^i) \tag{C.8}$$

Transport Properties

Zerol 60 oil:

$$\mu = -0.000122996T + 0.048002276 \tag{C.9}$$

with T in K, μ in Pa-s

$$k = 0.17 \text{ kW/m-K} \tag{C.10}$$

PAG 0-OB-1020 Oil (Booser, 1997):

$$\rho = -0.726923T + 1200.22 \tag{C.11}$$

Copeland 32-3MAF POE Oil:

$$\rho = (-0.00074351165(T - 273.15) + 0.9924395) * 1000 \tag{C.12}$$

$$\mu = 0.0002389593(\ln(T))^2 - 0.1927238779\ln(T) + 40.3718884485)\rho \cdot \text{1e-6} \tag{C.13}$$

with T in K, μ in Pa-s, ρ in kg/m^3

Table C.3 Coefficients for heat capacity, internal energy and entropy.

Fluid Type	c_0	c_1	c_2	c_3	Source
Water	5.10971	-0.00221776	-1.1713932x10^{-5}	2.967977x10^{-8}	Yaws (1999)
Zerol 60	0.337116	0.005186	0.0	0.0	Hugenroth (2006)
PAG 0-OB-1020 Oil	2.74374x10^{-3}	1.08646	0.0	0.0	Booser (1997)
PAO 40 Oil	1.08351	0.00352	0.0	0.0	Booser (1997)
POE Oil	2.30	0.0	0.0	0.0	Totten (2003)

C.5 Leakage Mass Flow Correction Terms

Derivation for Compressible Flow with Variable Area and Real Gas Properties

The analysis presented here largely follows that from Wassgren (2009), but modifications are made for the addition of changing area and real gas properties.

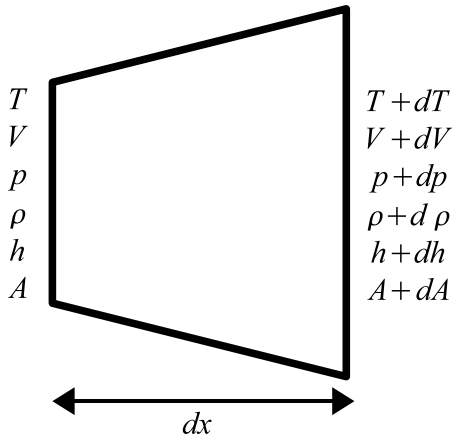


Figure C.2. Control volume for real gas analysis in leakage flow.

Assumptions:

- No heat transfer
- No mass transfer
- Variable area
- Compressible flow
- Real gas properties

Continuity:

$$\dot{m} = \rho V A \tag{C.14}$$

$$\frac{d\dot{m}}{dx} = 0 \tag{C.15}$$

$$\boxed{\frac{1}{\rho}\frac{d\rho}{dx} + \frac{1}{V}\frac{dV}{dx} + \frac{1}{A}\frac{dA}{dx} = 0} \tag{C.16}$$

MOMENTUM:

$$\int_{CS} u_x(\rho \mathbf{u}_{rel} \cdot d\mathbf{A}) = -\dot{m}V + (\rho + d\rho)(V + dV)^2(A + dA) \quad \text{(C.17)}$$

$$= -\dot{m}V + \dot{m}(V + dV) \quad \text{(C.18)}$$

$$= \dot{m}dV \quad \text{(C.19)}$$

since

$$\dot{m} = (\rho + d\rho)(V + dV)(A + dA) \quad \text{(C.20)}$$

The surface forces on the control volume are given by

$$F_{x,surface} = pA - (p + dp) \cdot (A + dA) - \tau_w P dx + (p + 1/2dp)dA \quad \text{(C.21)}$$

yields

$$\dot{m}dV = pA - (p + dp) \cdot (A + dA) - \tau_w P dx + (p + 1/2dp)dA \quad \text{(C.22)}$$

Drop products of differentials ($dAdp$) and divide through by dx

$$\boxed{\rho V \frac{dV}{dx} + \frac{dp}{dx} = -\frac{\rho V^2}{2} \frac{4 f_F}{D_H}} \quad \text{(C.23)}$$

The Fanning friction factor is given by

$$f_F = \begin{cases} \dfrac{24}{\mathrm{Re}} & \mathrm{Re} < 1736.5 \\ \dfrac{(0.790 \ln \mathrm{Re} - 1.64)^{-2}}{4} & \mathrm{Re} > 1736.5 \end{cases} \quad \text{(C.24)}$$

which assumes the flow can be treated like between infinite plates. In the transitional Reynolds number regime, the turbulent and laminar Reynolds number curves are extrapolated to intersection in order to ensure that the friction factor curve is monotonic, aiding numerical convergence. The Reynolds number is based on the local hydraulic diameter of the flow path, and is defined below for each flow path.

ENERGY:

$$\dot{m}\left(h + \frac{V^2}{2}\right) - \dot{m}\left(h + dh + \frac{(V + dV)^2}{2}\right) = 0 \quad \text{(C.25)}$$

$$\boxed{\frac{dh}{dx} + V\frac{dV}{dx} = 0} \quad \text{(C.26)}$$

Stagnation enthalpy is constant since no HT or boundary work

Entropy (Tds equation):

$$Tds - dh + vdp = 0 \tag{C.27}$$

$$\boxed{\frac{ds}{dx} - \frac{1}{T}\frac{dh}{dx} + \frac{1}{\rho T}\frac{dp}{dx} = 0} \tag{C.28}$$

p and h need to be expanded in terms of T and ρ

$$dh = \frac{\partial h}{\partial T}dT + \frac{\partial h}{\partial \rho}d\rho \tag{C.29}$$

$$dp = \frac{\partial p}{\partial T}dT + \frac{\partial p}{\partial \rho}d\rho \tag{C.30}$$

and dividing through by dx yields

$$\boxed{\frac{dh}{dx} = \frac{\partial h}{\partial T}\frac{dT}{dx} + \frac{\partial h}{\partial \rho}\frac{d\rho}{dx}} \tag{C.31}$$

$$\boxed{\frac{dp}{dx} = \frac{\partial p}{\partial T}\frac{dT}{dx} + \frac{\partial p}{\partial \rho}\frac{d\rho}{dx}} \tag{C.32}$$

Definition of Property Derivatives

The residual Helmholtz formulation of the properties gives the enthalpy in the form

$$\frac{h}{RT} = \tau\left[\left(\frac{\partial \alpha^0}{\partial \tau}\right)_\delta + \left(\frac{\partial \alpha^r}{\partial \tau}\right)_\delta\right] + \delta\left(\frac{\partial \alpha^r}{\partial \delta}\right)_\tau + 1 \tag{C.33}$$

where $\tau = T_c/T$ and $\delta = \rho/\rho_c$

$$\left(\frac{\partial h}{\partial \rho}\right)_T = \frac{\partial h}{\partial \delta}\frac{\partial \delta}{\partial \rho} = \frac{\partial h}{\partial \delta}\frac{1}{\rho_c} \tag{C.34}$$

$$\frac{\partial h}{\partial \delta} = RT\left\{\tau\left[\left(\frac{\partial^2 \alpha^0}{\partial \tau \partial \delta}\right)_\delta + \left(\frac{\partial^2 \alpha^r}{\partial \tau \partial \delta}\right)_\delta\right] + \left(\frac{\partial \alpha^r}{\partial \delta}\right)_\tau + \delta\left(\frac{\partial^2 \alpha^r}{\partial \delta^2}\right)_\tau\right\} \tag{C.35}$$

$$\frac{\partial h}{\partial \rho} = \frac{RT}{\rho_c}\left\{\tau\left[\left(\frac{\partial^2 \alpha^0}{\partial \tau \partial \delta}\right)_\delta + \left(\frac{\partial^2 \alpha^r}{\partial \tau \partial \delta}\right)_\delta\right] + \left(\frac{\partial \alpha^r}{\partial \delta}\right)_\tau + \delta\left(\frac{\partial^2 \alpha^r}{\partial \delta^2}\right)_\tau\right\} \tag{C.36}$$

$$\frac{\partial h}{\partial T} = \left(\frac{\partial h}{\partial T}\right)_\rho + \frac{\partial h}{\partial \tau}\frac{\partial \tau}{\partial T} = \left(\frac{\partial h}{\partial T}\right)_\rho + \frac{\partial h}{\partial \tau}\left(\frac{-T_c}{T^2}\right) \tag{C.37}$$

$$\left(\frac{\partial h}{\partial T}\right)_\rho = R\tau\left[\left(\frac{\partial \alpha^0}{\partial \tau}\right)_\delta + \left(\frac{\partial \alpha^r}{\partial \tau}\right)_\delta\right] + R\delta\left(\frac{\partial \alpha^r}{\partial \delta}\right)_\tau + R \tag{C.38}$$

$$\frac{\partial h}{\partial \tau} = RT\left[\left(\frac{\partial \alpha^0}{\partial \tau}\right)_\delta + \left(\frac{\partial \alpha^r}{\partial \tau}\right)_\delta\right] + RT\tau\left[\left(\frac{\partial^2 \alpha^0}{\partial \tau^2}\right)_\delta + \left(\frac{\partial^2 \alpha^r}{\partial \tau^2}\right)_\delta\right] + RT\delta\left(\frac{\partial^2 \alpha^r}{\partial \delta \partial \tau}\right)_\tau \tag{C.39}$$

Lemmon (2000) gives the derivatives for pressure directly:

$$\left(\frac{\partial p}{\partial \rho}\right)_T = RT\left[1 + 2\delta\left(\frac{\partial \alpha^r}{\partial \delta}\right)_\tau + \delta^2\left(\frac{\partial^2 \alpha^r}{\partial \delta^2}\right)_\tau\right] \tag{C.40}$$

$$\left(\frac{\partial p}{\partial T}\right)_\rho = R\rho\left[1 + \delta\left(\frac{\partial \alpha^r}{\partial \delta}\right)_\tau - \delta\tau\left(\frac{\partial^2 \alpha^r}{\partial \delta \partial \tau}\right)\right] \tag{C.41}$$

Solution for System of ODE

Using a computer algebra system (Maxima), the solution for the system of differential equations is obtained:

$$\frac{dT}{dx} = -\frac{2\,\rho\,A\,f_F\,V^4 + \left(\frac{dA}{dx}\left(\frac{\partial p}{\partial \rho}\,\rho\,D_H - \frac{\partial h}{\partial \rho}\,\rho^2\,D_H\right) - 2\,\frac{\partial h}{\partial \rho}\,\rho^2\,A\,f_F\right)V^2}{A\left(\rho\,D_H\,\frac{\partial h}{\partial T} - D_H\,\frac{\partial p}{\partial T}\right)V^2 + \rho\,A\,D_H\left(\frac{\partial h}{\partial \rho}\,\frac{\partial p}{\partial T} - \frac{\partial p}{\partial \rho}\,\frac{\partial h}{\partial T}\right)} \tag{C.42}$$

$$\frac{dp}{dx} = -\frac{2\,\rho\,A\,\frac{\partial p}{\partial T}\,f_F\,V^4 + \rho^2\left(2Af_F + \frac{dA}{dx}\,D_H\right)\left(\frac{\partial p}{\partial \rho}\,\frac{\partial h}{\partial T} - \frac{\partial h}{\partial \rho}\,\frac{\partial p}{\partial T}\right)V^2}{A\left(\rho\,D_H\,\frac{\partial h}{\partial T} - D_H\,\frac{\partial p}{\partial T}\right)V^2 + \rho\,A\,D_H\left(\frac{\partial h}{\partial \rho}\,\frac{\partial p}{\partial T} - \frac{\partial p}{\partial \rho}\,\frac{\partial h}{\partial T}\right)} \tag{C.43}$$

$$\frac{d\rho}{dx} = -\frac{\left(2\,\rho^2\,A\,\frac{\partial h}{\partial T}\,f_F + \frac{dA}{dx}\left(\rho^2\,D_H\,\frac{\partial h}{\partial T} - \rho\,D_H\,\frac{\partial p}{\partial T}\right)\right)V^2}{A\left(\rho\,D_H\,\frac{\partial h}{\partial T} - D_H\,\frac{\partial p}{\partial T}\right)V^2 + \rho\,A\,D_H\left(\frac{\partial h}{\partial \rho}\,\frac{\partial p}{\partial T} - \frac{\partial p}{\partial \rho}\,\frac{\partial h}{\partial T}\right)} \tag{C.44}$$

$$\frac{dh}{dx} = -\frac{2\,\rho\,A\,\dfrac{\partial h}{\partial T}\,f_F\,V^4 + \rho\,\dfrac{dA}{dx}\,D_H\,\left(\dfrac{\partial p}{\partial \rho}\,\dfrac{\partial h}{\partial T} - \dfrac{\partial h}{\partial \rho}\,\dfrac{\partial p}{\partial T}\right)\,V^2}{A\,\left(\rho\,D_H\,\dfrac{\partial h}{\partial T} - D_H\,\dfrac{\partial p}{\partial T}\right)\,V^2 + \rho\,A\,D_H\,\left(\dfrac{\partial h}{\partial \rho}\,\dfrac{\partial p}{\partial T} - \dfrac{\partial p}{\partial \rho}\,\dfrac{\partial h}{\partial T}\right)} \tag{C.45}$$

$$\frac{dV}{dx} = \frac{2\,\rho\,A\,\dfrac{\partial h}{\partial T}\,f_F\,V^3 + \rho\,\dfrac{dA}{dx}\,D_H\,\left(\dfrac{\partial p}{\partial \rho}\,\dfrac{\partial h}{\partial T} - \dfrac{\partial h}{\partial \rho}\,\dfrac{\partial p}{\partial T}\right)\,V}{A\,\left(\rho\,D_H\,\dfrac{\partial h}{\partial T} - D_H\,\dfrac{\partial p}{\partial T}\right)\,V^2 + \rho\,A\,D_H\,\left(\dfrac{\partial h}{\partial \rho}\,\dfrac{\partial p}{\partial T} - \dfrac{\partial p}{\partial \rho}\,\dfrac{\partial h}{\partial T}\right)} \tag{C.46}$$

$$\frac{ds}{dx} = -\frac{2\,f_F\,V^2}{D_H\,T} \tag{C.47}$$

This system of equations are then integrated from x_1 to x_2 in order to determine all the properties along the flow path. If the inlet flow velocity is not known, the inlet velocity must be iteratively determined to match the outlet of the flow path.

Radial flow path

For the radial flow path, the geometry can be given by that shown in Fig. (C.3). The flow is outwards through a cylindrical section with height δ. The inner and outer radii are set to be equal to radii of curvature of the inner and outer scroll wraps.

For the isentropic compressible nozzle flow model, the upstream and downstream pressures and the throat area are given. This model assumes that the fluid is a perfect gas with constant specific heats with compressibility taken into account, but there is no friction. If the imposed pressure ratio is large enough to obtain sonic conditions at the throat of the nozzle, the flow is choked. The pressure ratio employed is given by

$$p_r = \begin{cases} \left(1 + \frac{(k-1)}{2}\right)^{k/(1-k)} & p_{down}/p_{up} <= \left(1 + \frac{(k-1)}{2}\right)^{k/(1-k)} \\ p_{down}/p_{up} & p_{down}/p_{up} > \left(1 + \frac{(k-1)}{2}\right)^{k/(1-k)} \end{cases} \tag{C.48}$$

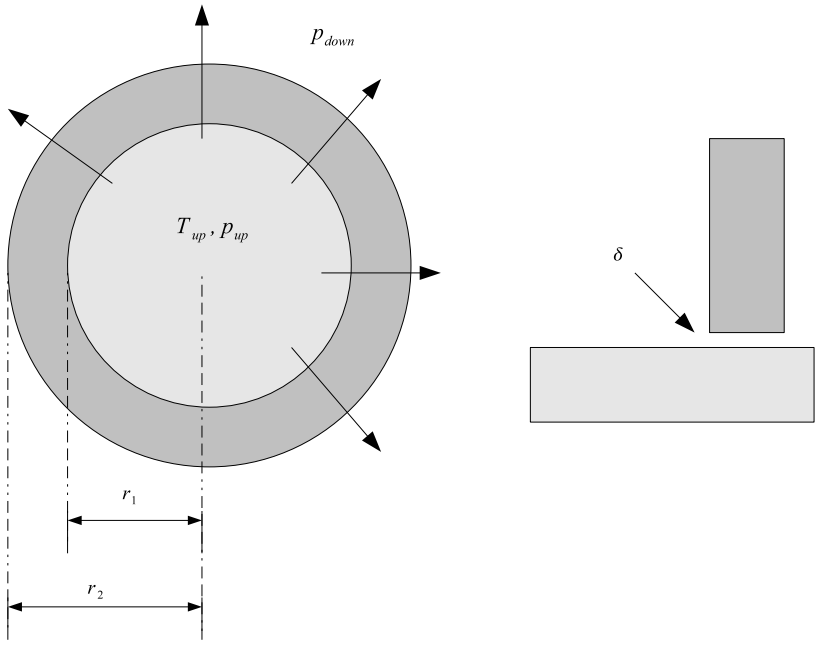


Figure C.3. Radial flow geometry schematic.

where k is the ratio of specific heats, given by $k = c_p/c_v$, evaluated at the upstream condition. Thus the mass flow rate is given by

$$\dot{m}_{nozzle} = A_{th}\sqrt{p_{up}\rho_{up}}\sqrt{\frac{2k}{(k-1)}\left(p_r^{2/k} - p_r^{(k+1)/k}\right)} \tag{C.49}$$

where the area A_{th} is the upstream area equal to

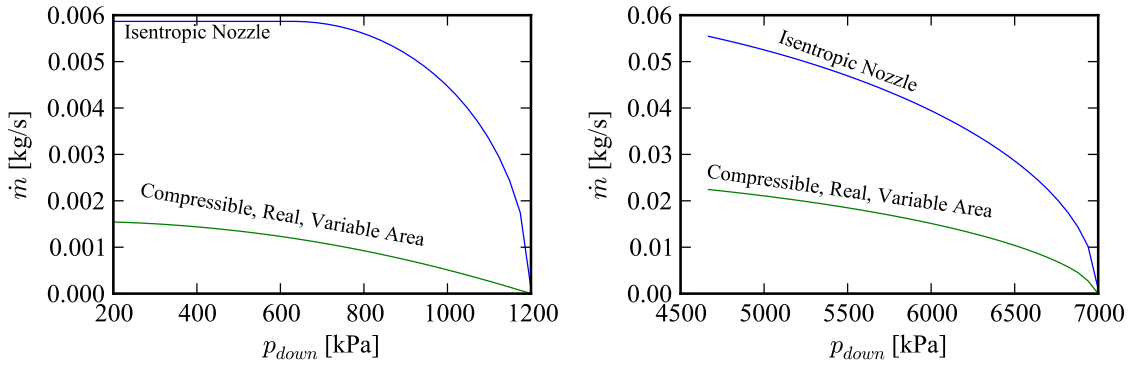
$$A_{th} = 2\pi r_1 \delta_{radial} \tag{C.50}$$

The Reynolds number for this flow can be obtained by using the definition

$$\mathrm{Re} = \frac{\frac{\dot{m}}{A_{th}} D_H}{\mu} \tag{C.51}$$

where the hydraulic diameter used is $D_H = 2\delta$ and the viscosity μ is given by the upstream state.

For the compressible frictional flow with variable area and real gas properties derived in this section, the inlet mass flow rate/velocity is not known, and must be



(a) Nitrogen (p_{up}=1200 kPa, T_{up}=320 K)

(b) CO_2 (p_{up}=7000 kPa, T_{up}=320 K)

Figure C.4. Nitrogen and CO_2 flow rates through the radial gap predicted by isentropic nozzle and detailed models ($\delta = 10\mu m$).

iteratively calculated. A numerical solver is used to enforce the downstream pressure to be equal to the imposed downstream pressure.

For the variable area model, the flow area is given by

$$A(x) = 2\pi x\delta \tag{C.52}$$

where x takes on the values in the range r_1 to r_2.

Two different flow configurations are considered, nitrogen and carbon dioxide with high-side pressures that might be experienced in a scroll compressor. This yields the flows shown in Figure C.4

While the isentropic nozzle model effectively captures the compressibility effects, it does not capture the frictional effects. It was therefore considered to correlate the ratio mass flow rates predicted by the isentropic nozzle flow model to that of the detailed frictional flow model with real gas properties and variable area. The same geometry was used for a range of fluids, detailed in Table C.4. For each refrigerant, the high-side pressure was varied through the range shown, and then for each upstream pressure, the downstream pressure was decreased until a pressure ratio of $p_{r,max}$ was reached.

Table C.4 Refrigerant states for development of frictional correction factor for radial leakage flow path.

Refrigerant	p_{up}	$p_{r,max}$	T_{up}
-	kPa	-	K
Nitrogen	1800-400	1.5	320
CO_2	8000-6000	1.5	320
R134a	1500-400	1.5	350
R410A	1500-1000	1.5	350

The geometry employed for each refrigerant is given in Table C.5. The outer radius r_2 can be found from

$$r_2 = r_1 + t \tag{C.53}$$

Table C.5 Geometric parameters for radial gap width calculation.

Parameter	Range
δ_{radial}	5-25 μm
t	2-10 mm
r_1	15.85 - 53.89 mm

From these calculations, the ratio of nozzle to detailed model flow rates can then be obtained, and a correlation for the mass flow ratio can be obtained. Figure C.5 shows the mass flow ratios for all the data points obtained, where the mass flow ratio is defined by

$$M \equiv \frac{\dot{m}_{nozzle}}{\dot{m}_{real}} \tag{C.54}$$

For a given set of parameters, the ratio of the isentropic nozzle mass flow rate prediction to that of the detailed model can be given by a correlation of the form

$$M = \frac{a_0(L/L_0)^{a_1}}{a_2(\delta/\delta_0) + a_3}[\xi(a_4 \mathrm{Re}^{a_5} + a_6) + (1-\xi)(a_7 \mathrm{Re}^{a_8} + a_9)] + a_{10} \tag{C.55}$$

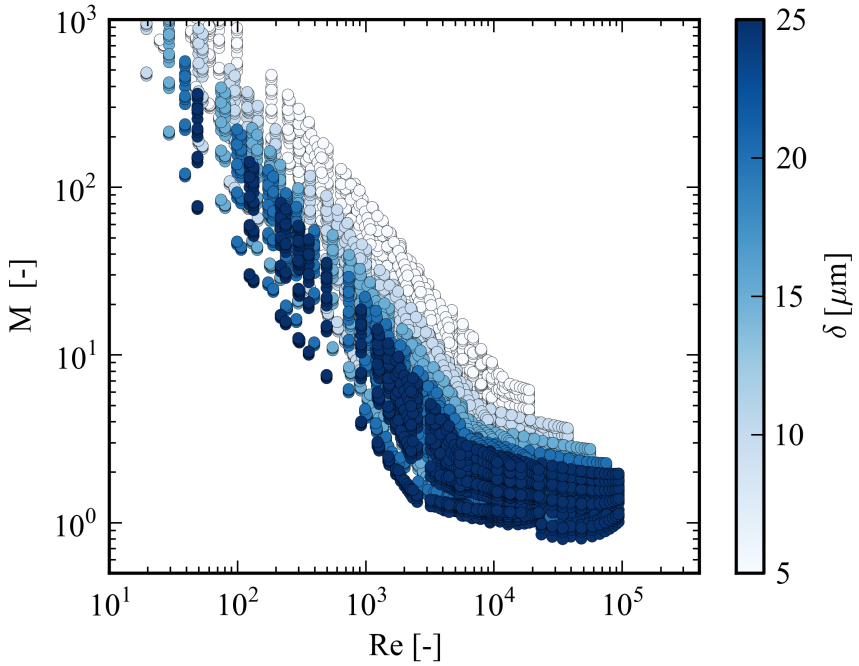


Figure C.5. Ratio of prediction of mass flow rate from isentropic nozzle model and prediction of mass flow rate from full detailed flow model for the radial leakage.

where the cross-over term ξ is given by

$$\xi = \frac{1}{1 + \exp[-0.01(\mathrm{Re} - \mathrm{Re}^{\star})]} \tag{C.56}$$

which is used to allow the curve fit optimizer to obtain two separate and continuous solutions for the low- and high-Reynolds numbers portions. The length in the correlation is equal to r_2-r_1, which for a scroll compressor is simply equal to the scroll wrap thickness t. The non-dimensionalization parameters L_0 and δ_0 are given by the values

$$\delta_0 = 10\ \mu m \tag{C.57}$$

$$L_0 = 0.005\ m \tag{C.58}$$

The necessary constants for the radial leakage flow path are found in Table C.6. MAE is 10.798%, and RMS error is 16.36. 12,000 points were used to develop the correlation. The error is plotted in Figure C.6

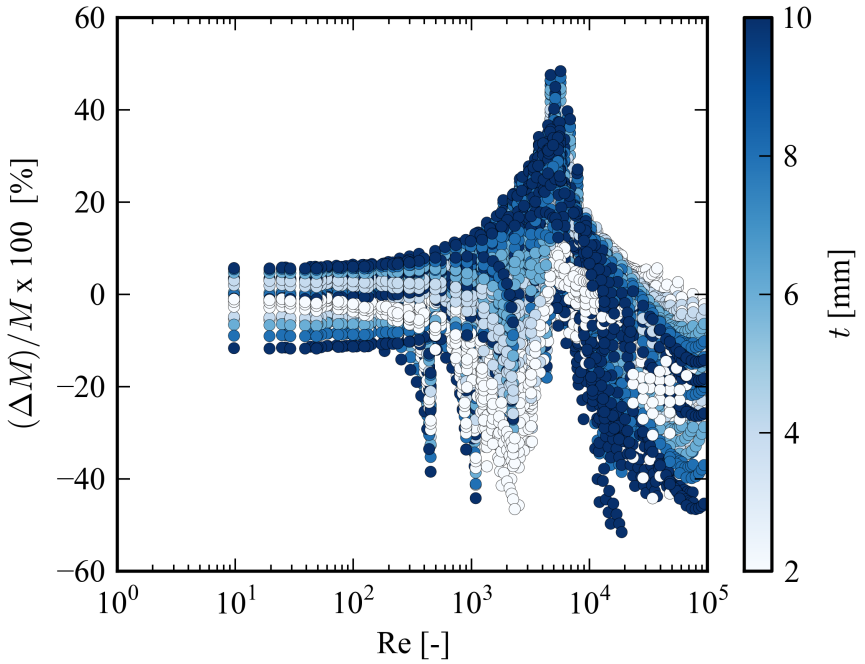


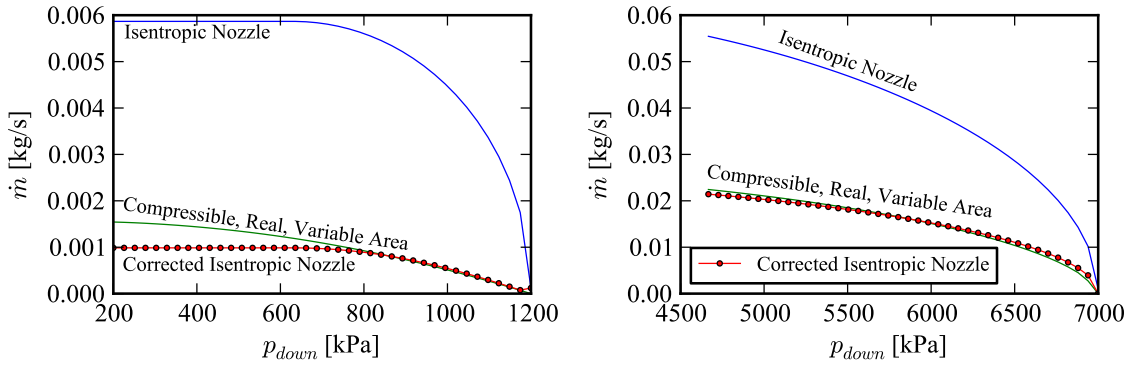
Figure C.6. Error of prediction of mass flow rate from isentropic nozzle model and prediction of mass flow rate from full detailed flow model for the radial leakage.

Table C.6 Coefficients for empirical correction term for radial leakage gap.

Coefficient	Value	Coefficient	Value
a_0	2.59321070e+04	a_6	-1.28861161e-02
a_1	9.14825434e-01	a_7	-1.51202604e+02
a_2	-1.77588568e+02	a_8	-9.99674458e-01
a_3	-2.37052788e-01	a_9	1.61435039e-02
a_4	-1.72347611e+05	a_{10}	8.25533457e-01
a_5	-1.20687600e+01	Re^*	5.24358195e+03

The corrected radial mass flow rate can then be obtained from

$$\dot{m}_{corr} = \frac{\dot{m}_{nozzle}}{M} \tag{C.59}$$



(a) Nitrogen (p_{up}=1200 kPa, T_{up}=320 K) (b) CO_2 (p_{up}=7000 kPa, T_{up}=320 K)

Figure C.7. Nitrogen and CO_2 flow rates through the radial gap predicted by isentropic nozzle, detailed models, and corrected isentropic nozzle.

The results for Nitrogen and CO_2 are shown in Fig. C.7. The correction term works extremely well for CO_2 since the isentropic nozzle model does not predict choking will occur over the range of back pressures investigated. For nitrogen, the isentropic nozzle model predicts choking will occur, and for those points where choking does occur, the corrected model underpredicts the detailed model. In practice, the flow in the scroll compressor is believed to be mostly not choked, so the correction should work quite well.

Flank Flow Path

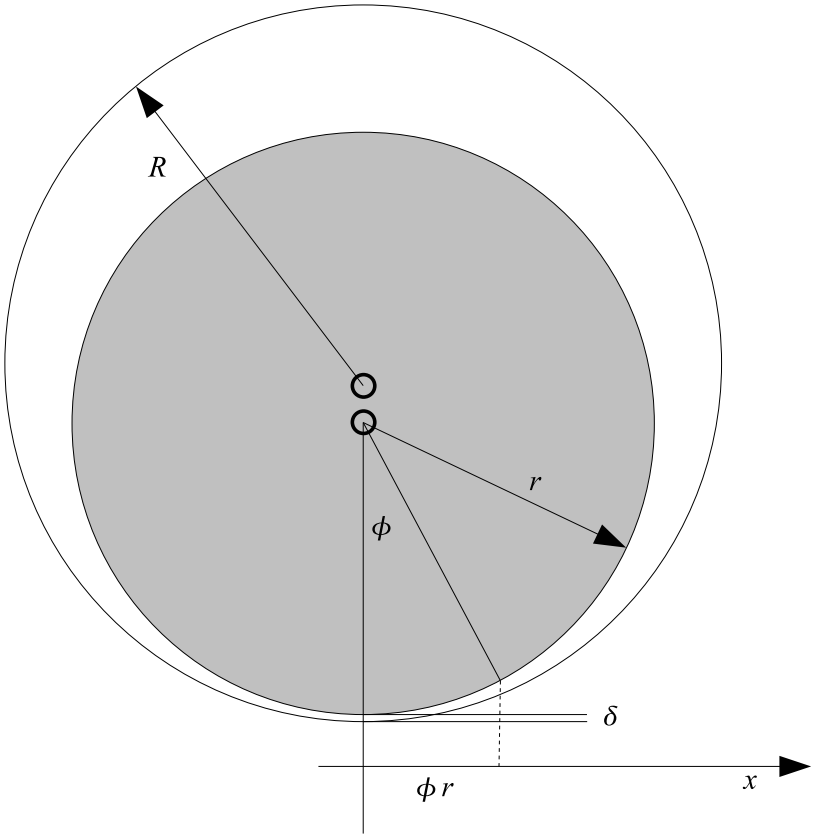


Figure C.8. Flank flow schematic.

For the flank flow path, the geometry is simplified to be equivalent to the flow between two conformal cylinders with the same height as shown in Fig. C.8. The area of the flow path is given as a function of the angle ϕ, which is the angle on the inner cylinder with radius r (Yanagisawa and Shimizu, 1985b):

$$A = h_s \left[R - (R - r - \delta)\cos\phi - \sqrt{r^2 - (R - r - \delta)^2 \sin^2\phi} \right] \tag{C.60}$$

where δ is the minimum gap width, and r and R are the radii of the inner and outer cylinders respectively. The x-coordinate corresponding to an angle ϕ is given by $x = \phi r$, so the derivative of the angle with respect to x is

$$\frac{d\phi}{dx} = \frac{1}{r} \tag{C.61}$$

and by the chain rule the derivative of the area with respect to x is given by

$$\frac{dA}{dx} = \frac{h_s}{r}\left[(R-r-\delta)\sin\phi + \frac{(R-r-\delta)^2\sin\phi\cos\phi}{\sqrt{r^2-(R-r-\delta)^2\sin^2\phi}}\right] \tag{C.62}$$

Therefore the analysis derived above for real gas properties, variable area and frictional flow can be used with the flank area relationships shown from Eqns. (C.60) and (C.62). The relationships are integrated along the flank flow path in order to obtain the pressure drop. The inlet Mach number (or alternatively mass flow rate or inlet velocity) is iteratively determined using a numerical solver in order to enforce the downstream pressure. Again the isentropic nozzle model is calculated, and then corrected with an empirical term. The throat area for the isentropic nozzle model is equal to

$$A_{th} = h_s\delta \tag{C.63}$$

Table C.7 Range of geometric parameters investigated for flank gap width calculation.

Parameter	Range
δ_{flank}	5-25 μm
L	2-10 mm
r	15.85 - 53.89 mm
h	32.89 mm

Since the shape of the mass flow curves for the flank leakages are analogous to those of the radial leakages, results will not be shown here. The geometric parameters employed can be found in Table C.7. A correction term of the form of Eqn. (C.55) is built for a range of gap widths with constants in Table C.8. The length term L is given by $R-r$ which simple algebra shows is equal to the orbiting radius r_o for the scroll compressor. The results for the flank gap width can be found in Figure C.9. MAE is 14.54% and RMS is 2.25. 5916 runs were used to develop the correlation. The error is shown in Figure C.10.

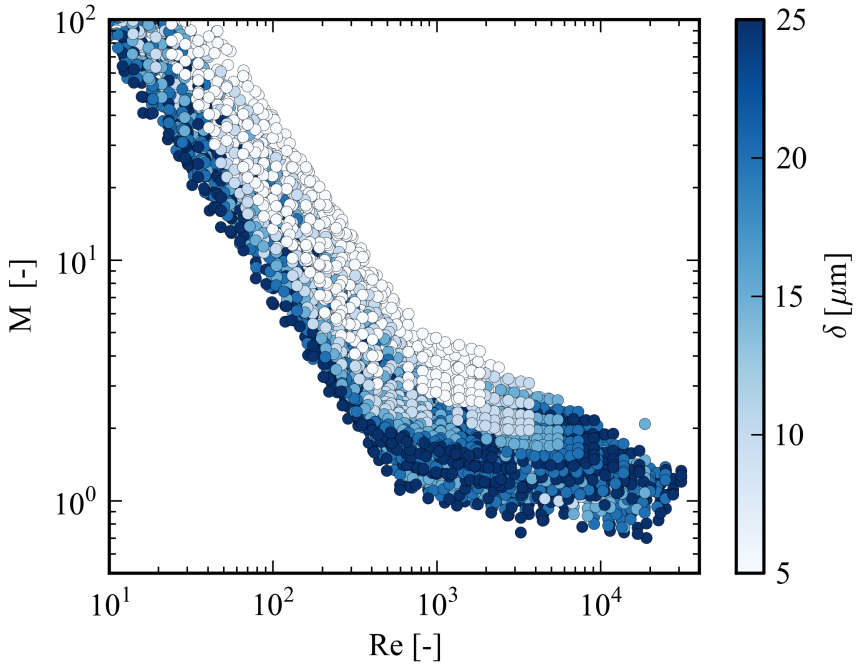


Figure C.9. Ratio of prediction of mass flow rate from isentropic nozzle model and prediction of mass flow rate from full detailed flow model for the flank leakage.

Table C.8 Coefficients for empirical correction term for flank flow path.

Coefficient	Value	Coefficient	Value
a_0	-2.63970396e+00	a_6	-5.10200923e-01
a_1	-5.67164431e-01	a_7	-1.20517483e+03
a_2	8.36554999e-01	a_8	-1.02938914e+00
a_3	8.10567168e-01	a_9	6.89497786e-01
a_4	6.17402826e+03	a_{10}	1.09607735e+00
a_5	-7.60907962e+00	Re*	8.26167178e+02

C.6 Summary

In order to apply this method, the following procedure is employed

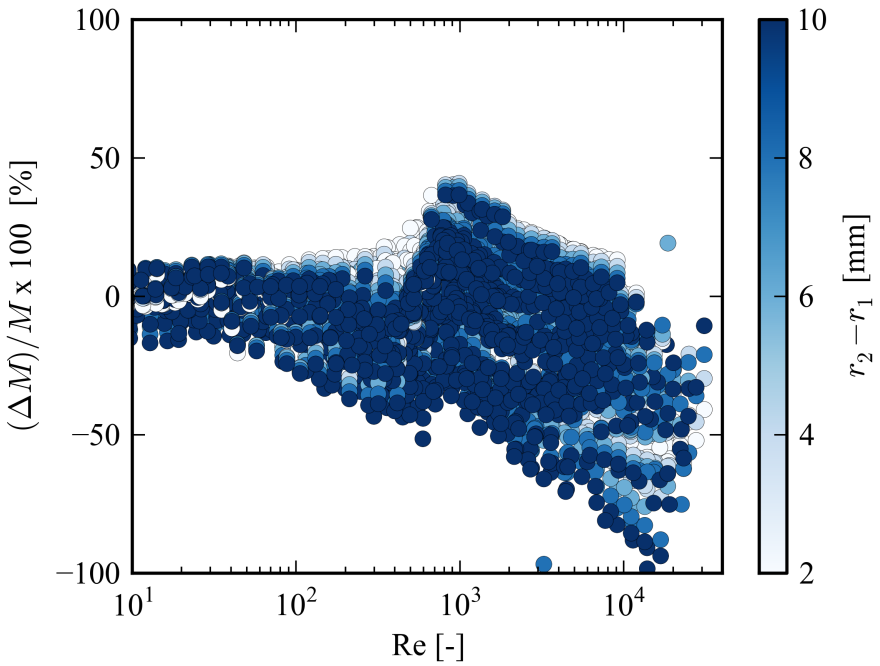


Figure C.10. Error of prediction of mass flow rate from isentropic nozzle model and prediction of mass flow rate from full detailed flow model for the flank leakage.

- Evaluate isentropic nozzle prediction for gas flow rate based on minimum area for flank leakage, and upstream area for radial leakage
- Use calculated flow rate to find the Reynolds number
- Evaluate the correlation for mass flow ratio with the constants appropriate to the flow path of interest
- Divide nozzle prediction of gas flow rate by correction term

C.7 Solution For Set Of ODE With Temperature And Pressure As State Variables

Conservation of Mass

The conservation of mass for a given control volume is expressed as

$$\frac{dm_{CV}}{d\theta} = \frac{1}{\omega}\sum \dot{m} \tag{C.64}$$

The mass of fluid contained in a given control volume is equal to the product of

$$m_{CV} = \rho_{CV} V_{CV} \tag{C.65}$$

where the differential of mass is therefore given by

$$dm_{CV} = d\rho_{CV} V_{CV} + \rho_{CV} dV_{CV} \tag{C.66}$$

Expressing the differential of mass of the control volume's contents in terms of crank angle θ yields

$$\frac{dm_{CV}}{d\theta} = \frac{d\rho_{CV}}{d\theta} V_{CV} + \frac{dV_{CV}}{d\theta}\rho_{CV} \tag{C.67}$$

But density is a function of temperature, pressure, and mass fraction, so the differential of density (expressed in terms of crank angle) is given by

$$\frac{d\rho_{CV}}{d\theta} = \frac{\partial \rho_{CV}}{\partial P}\frac{dP}{d\theta} + \frac{\partial \rho_{CV}}{\partial T}\frac{dT}{d\theta} + \frac{\partial \rho_{CV}}{\partial x_l}\frac{dx_l}{d\theta} \tag{C.68}$$

Substitution of Eqn. (C.68) into Eqn. (C.67) yields

$$\frac{dm_{CV}}{d\theta} = \left(\frac{\partial \rho_{CV}}{\partial P}\frac{dP}{d\theta} + \frac{\partial \rho_{CV}}{\partial T}\frac{dT}{d\theta} + \frac{\partial \rho_{CV}}{\partial x_l}\frac{dx_l}{d\theta}\right) V_{CV} + \frac{dV_{CV}}{d\theta}\rho_{CV} \tag{C.69}$$

And thus conservation of mass for the control volume is expressed as

$$\left(\frac{\partial \rho_{CV}}{\partial P}\frac{dP}{d\theta} + \frac{\partial \rho_{CV}}{\partial T}\frac{dT}{d\theta} + \frac{\partial \rho_{CV}}{\partial x_l}\frac{dx_l}{d\theta}\right) V_{CV} + \frac{dV_{CV}}{d\theta}\rho_{CV} = \frac{1}{\omega}\sum \dot{m} \tag{C.70}$$

Rearrangement and grouping in terms of the unknown derivatives yields

$$\left(\frac{\partial \rho_{CV}}{\partial T} V_{CV}\right)\frac{dT}{d\theta} + \left(\frac{\partial \rho_{CV}}{\partial P} V_{CV}\right)\frac{dP}{d\theta} + \left(\frac{\partial \rho_{CV}}{\partial x_l} V_{CV}\right)\frac{dx_l}{d\theta} = \frac{1}{\omega}\sum \dot{m} - \frac{dV_{CV}}{d\theta}\rho_{CV} \tag{C.71}$$

Conservation of Energy

The treatment of conservation of energy is similar to that of conservation of mass. With the assumptions of negligible kinetic and potential energy of the control volume, conservation of energy is given by

$$\frac{dU_{CV}}{d\theta} = \frac{\dot{Q}}{\omega} + \frac{\dot{W}}{\omega} + \frac{1}{\omega}\sum \dot{m}h \tag{C.72}$$

but since $\dot{W} = -P\frac{dV}{dt}$, conservation of energy can be expressed as

$$\frac{dU_{CV}}{d\theta} = \frac{\dot{Q}}{\omega} - P\frac{dV}{d\theta} + \frac{1}{\omega}\sum \dot{m}h \tag{C.73}$$

and the derivative of specific internal energy with respect to the crank angle is given by

$$\frac{du_{CV}}{d\theta} = \frac{\partial u_{CV}}{\partial P}\frac{dP}{d\theta} + \frac{\partial u_{CV}}{\partial T}\frac{dT}{d\theta} + \frac{\partial u_{CV}}{\partial x_l}\frac{dx_l}{d\theta} \tag{C.74}$$

Expansion of the internal energy term yields

$$\frac{dU_{CV}}{d\theta} = \frac{dm_{CV}}{d\theta}u_{CV} + \frac{du_{CV}}{d\theta}m_{CV} \tag{C.75}$$

which with substitution yields

$$\frac{dU_{CV}}{d\theta} = \frac{dm_{CV}}{d\theta}u_{CV} + \left(\frac{\partial u_{CV}}{\partial P}\frac{dP}{d\theta} + \frac{\partial u_{CV}}{\partial T}\frac{dT}{d\theta} + \frac{\partial u_{CV}}{\partial x_l}\frac{dx_l}{d\theta}\right)m_{CV} \tag{C.76}$$

The derivative of mass in the control volume is obtained from Eqn. (C.64), the derivative of specific internal energy from Eqn. (C.74) and thus the conservation of energy is expressed as

$$\begin{aligned}&\left(\frac{1}{\omega}\sum \dot{m}\right)u_{CV} + \left(\frac{\partial u_{CV}}{\partial P}\frac{dP}{d\theta} + \frac{\partial u_{CV}}{\partial T}\frac{dT}{d\theta} + \frac{\partial u_{CV}}{\partial x_l}\frac{dx_l}{d\theta}\right)m_{CV}\\ &= \frac{\dot{Q}}{\omega} - P\frac{dV}{d\theta} + \frac{1}{\omega}\sum \dot{m}h\end{aligned} \tag{C.77}$$

Which with grouping for the unknown derivatives yields

$$\begin{aligned}&\left(\frac{\partial u_{CV}}{\partial T}m_{CV}\right)\frac{dT}{d\theta} + \left(\frac{\partial u_{CV}}{\partial P}m_{CV}\right)\frac{dP}{d\theta} + \left(\frac{\partial u_{CV}}{\partial x_l}m_{CV}\right)\frac{dx_l}{d\theta}\\ &= \frac{\dot{Q}}{\omega} - P\frac{dV}{d\theta} + \frac{1}{\omega}\sum \dot{m}h + \left(\frac{1}{\omega}\sum \dot{m}\right)u_{CV}\end{aligned} \tag{C.78}$$

Conservation of Oil Mass

Finally, the oil mass can be treated in a similar manner where the conservation of oil mass in a given control volume is expressed as

$$\frac{dm_l}{d\theta} = \frac{1}{\omega}\sum \dot{m} x_{l,f} \tag{C.79}$$

where the same sign convention on the mass flow is used, and $x_{l,f}$ is the upstream mass fraction of the flow path. For the control volume

$$\frac{dm_l}{d\theta} = x_l \frac{dm_{CV}}{d\theta} + m_{CV}\frac{dx_l}{d\theta} \tag{C.80}$$

Thus with substitution of Eqn. (C.80), (C.64) and (C.79), the conservation of oil mass is given by

$$m_{CV}\frac{dx_l}{d\theta} = \frac{1}{\omega}\sum \dot{m} x_{l,f} - x_l\left(\frac{1}{\omega}\sum \dot{m}\right) \tag{C.81}$$

Solution

Finally, to solve for the property derivatives, the following linear system is solved:

$$\mathbf{A}\begin{bmatrix} \dfrac{dT}{d\theta} \\ \dfrac{dP}{d\theta} \\ \dfrac{dx_l}{d\theta} \end{bmatrix} = \mathbf{b} \tag{C.82}$$

where

$$\mathbf{A} = \begin{bmatrix} \frac{\partial \rho_{CV}}{\partial T}V_{CV} & \frac{\partial \rho_{CV}}{\partial P}V_{CV} & \frac{\partial \rho_{CV}}{\partial x_l}V_{CV} \\ \frac{\partial u_{CV}}{\partial T}m_{CV} & \frac{\partial u_{CV}}{\partial P}m_{CV} & \frac{\partial u_{CV}}{\partial x_l}m_{CV} \\ 0 & 0 & m_{CV} \end{bmatrix} \tag{C.83}$$

$$\mathbf{b} = \begin{bmatrix} \frac{1}{\omega}\sum \dot{m} - \dfrac{dV_{CV}}{d\theta}\rho_{CV} \\ \frac{\dot{Q}}{\omega} - P\dfrac{dV}{d\theta} + \frac{1}{\omega}\sum \dot{m}h + \left(\frac{1}{\omega}\sum \dot{m}\right)u_{CV} \\ \frac{1}{\omega}\sum \dot{m} x_{l,f} - x_l\left(\frac{1}{\omega}\sum \dot{m}\right) \end{bmatrix} \tag{C.84}$$

which yields the solution

$$\frac{dT}{d\theta} = -\frac{-\rho^2 \frac{du}{dP}\frac{dV}{d\theta} + \frac{d\rho}{dP}P\frac{dV}{d\theta} - \frac{d\rho}{dP}\frac{\dot{Q}}{\omega} + \sum\left[\frac{\dot{m}}{\omega}\left(\rho\frac{du_{CV}}{dP} + \frac{d\rho}{dP}\left(u_{CV}-h\right) + \left(\frac{d\rho}{dP}\frac{du}{dx_l} - \frac{du}{dP}\frac{d\rho}{dx_l}\right)\left(x_{l,f}-x_l\right)\right)\right]}{\rho V\left(\frac{du}{dT}\frac{d\rho}{dP} - \frac{du}{dP}\frac{d\rho}{dT}\right)}$$

$$\frac{dP}{d\theta} = \frac{-\rho^2 \frac{du}{dT}\frac{dV}{d\theta} + \frac{d\rho}{dT}P\frac{dV}{d\theta} - \frac{d\rho}{dT}\frac{\dot{Q}}{\omega} + \sum\left[\frac{\dot{m}}{\omega}\left(\rho\frac{du_{CV}}{dT} + \frac{d\rho}{dT}\left(u_{CV}-h\right) + \left(\frac{d\rho}{dT}\frac{du}{dx_l} - \frac{du}{dT}\frac{d\rho}{dx_l}\right)\left(x_{l,f}-x_l\right)\right)\right]}{\rho V\left(\frac{du}{dT}\frac{d\rho}{dP} - \frac{du}{dP}\frac{d\rho}{dT}\right)}$$

$$\frac{dx_l}{d\theta} = -\frac{\sum\left[\frac{\dot{m}}{\omega}\left(x_l - x_{l,f}\right)\right]}{\rho V} \tag{C.85}$$

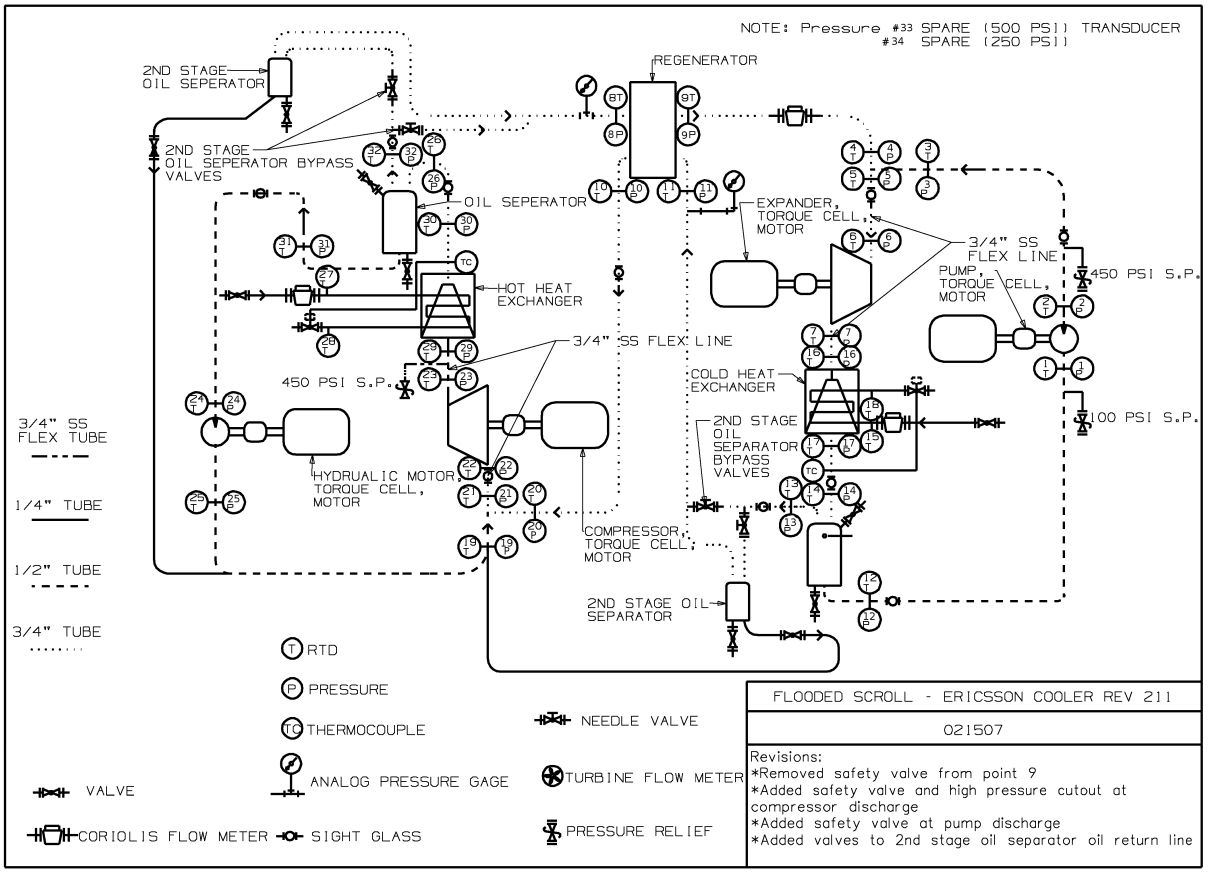


Figure D.1. Detailed Liquid-Flooded Ericsson Cycle Layout.

Table D.1: Validation Data for Compressor Model.

Run	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}
-	K	K	K	K	K	K	K	K	K	K	K
1	289.5	290.4	290.1	292.7	290.5	290.4	287.4	308.2	293.0	304.3	289.7
2	288.4	289.7	289.2	291.8	289.8	289.8	282.8	306.3	292.0	302.6	288.7
3	288.4	289.9	289.4	291.0	290.0	290.1	276.2	303.0	291.1	300.0	288.3
4	288.3	289.3	289.0	293.1	289.9	289.8	285.2	314.2	293.5	308.3	288.4
5	288.3	289.6	289.2	292.8	290.2	290.4	281.0	312.5	293.2	307.0	288.2
6	288.5	290.3	289.9	292.0	290.7	291.1	274.4	308.2	292.3	303.7	288.1
7	289.1	290.0	289.7	295.0	291.1	291.0	285.5	320.9	295.9	313.1	288.8
8	289.6	290.8	290.5	295.1	292.1	292.4	282.1	318.5	295.5	311.4	289.3
9	289.4	291.5	291.2	294.2	292.5	293.1	275.8	313.9	294.5	307.9	289.1
10	289.7	290.6	290.4	293.4	290.8	290.8	287.6	310.0	293.8	305.9	290.0
11	289.2	290.5	290.0	292.7	290.7	290.8	284.0	307.8	293.1	304.0	289.5
12	287.8	290.1	289.5	290.7	289.9	290.2	276.7	302.1	290.8	299.3	287.8
13	293.6	294.7	294.4	299.2	295.6	295.5	290.5	323.5	299.4	316.5	293.5
14	289.8	291.2	290.8	294.9	292.2	292.5	283.0	317.0	295.2	310.6	289.6
15	287.4	290.8	290.3	291.8	290.8	291.2	274.5	310.0	291.7	304.7	287.1
16	291.3	292.6	292.2	298.2	293.8	293.8	288.1	325.7	298.6	317.4	291.2
17	291.6	293.2	292.8	297.7	294.7	295.2	284.8	323.1	298.3	315.3	291.1
18	284.6	290.9	290.8	290.7	290.3	290.6	273.0	316.7	291.3	308.8	283.6
19	299.1	299.6	299.7	301.5	300.0	299.7	296.3	320.1	302.2	315.5	298.4
20	299.7	300.3	300.4	302.4	300.9	300.7	293.8	318.9	303.0	314.8	299.5
21	300.2	300.4	300.7	302.3	301.3	301.2	287.5	314.8	302.8	311.6	300.1
22	300.5	301.2	301.2	304.4	301.9	301.6	297.3	326.0	305.1	320.1	300.1
23	300.9	301.7	301.7	304.8	302.7	302.6	294.0	324.2	305.3	318.9	300.8
24	300.6	301.3	301.7	303.9	302.6	302.8	286.7	319.2	304.3	315.0	300.7
25	300.9	301.7	301.7	306.2	302.9	302.7	297.4	331.4	307.0	323.9	300.6
26	301.3	302.1	302.3	306.5	303.9	304.0	293.9	329.4	307.0	322.5	301.2
27	300.4	302.0	302.4	305.1	304.0	304.2	286.3	324.4	305.6	318.6	300.5

Continued on next page

Table D.1: Continued.

Run	T_{12}	T_{13}	T_{14}	T_{15}	T_{16}	T_{17}	T_{18}	T_{19}	T_{20}	T_{21}	T_{22}
-	K	K	K	K	K	K	K	K	K	K	K
1	289.4	289.4	293.9	292.4	286.6	288.5	290.1	310.2	304.7	309.1	310.0
2	288.2	288.3	293.8	292.1	281.5	287.3	288.7	307.3	303.1	306.2	306.9
3	288.1	287.9	293.8	291.8	274.2	287.2	287.7	303.5	300.4	302.2	302.9
4	288.2	288.2	294.2	293.3	284.4	287.4	289.5	316.3	308.9	314.5	315.7
5	288.1	288.0	294.1	294.0	279.6	287.3	288.9	313.6	307.6	311.5	312.6
6	288.2	288.0	294.0	294.1	272.5	287.8	288.3	308.7	304.2	306.5	307.4
7	289.0	288.9	294.1	295.4	284.7	288.1	290.9	322.6	313.9	320.2	321.7
8	289.4	289.3	294.3	296.2	280.8	288.8	290.7	319.3	312.2	316.3	317.9
9	289.3	289.1	294.3	296.8	274.2	289.6	290.4	314.2	308.6	311.2	312.3
10	289.5	289.5	289.5	293.9	286.8	288.6	289.4	311.8	306.4	310.7	311.6
11	288.9	289.0	288.7	293.7	282.9	288.0	288.3	309.0	304.5	307.9	308.6
12	287.3	287.2	286.7	293.5	275.0	286.5	283.6	302.6	299.7	301.0	301.8
13	293.5	293.4	293.4	299.2	289.7	292.6	293.7	324.9	317.4	323.0	324.3
14	289.5	289.4	289.4	297.5	281.9	288.8	288.4	318.1	311.5	315.7	317.0
15	287.2	286.8	286.6	295.7	273.1	287.2	284.0	310.5	305.4	308.0	309.1
16	291.2	291.0	290.9	299.1	287.7	290.6	291.8	327.0	318.5	324.7	326.1
17	291.1	290.9	290.9	300.1	283.4	290.5	288.9	324.0	316.3	320.8	322.5
18	284.2	283.1	282.6	299.8	271.2	283.0	276.4	317.1	309.9	313.3	314.8
19	299.1	298.9	295.4	303.0	296.0	298.9	300.9	323.1	315.6	321.6	322.6
20	299.9	299.8	295.8	303.6	293.2	299.6	300.4	320.8	314.9	319.2	320.0
21	300.3	300.5	295.0	303.8	286.8	300.6	300.0	315.7	311.7	314.0	314.8
22	300.6	300.5	296.2	304.9	296.8	300.1	301.8	328.5	320.5	326.6	327.8
23	301.2	301.1	296.2	305.5	293.4	300.9	301.7	325.7	319.2	323.6	324.7
24	300.8	301.0	295.2	305.5	285.9	301.1	301.4	319.9	315.2	317.6	318.5
25	301.0	301.0	296.5	306.0	296.9	300.4	302.5	333.9	324.5	331.4	332.9
26	301.5	301.6	296.4	306.7	293.2	301.6	302.5	330.6	323.1	327.6	329.1
27	300.6	300.7	295.5	306.9	285.6	300.8	301.7	324.9	319.1	321.9	322.9

Continued on next page

Table D.1: Continued.

Run	T_{23}	T_{24}	T_{25}	T_{26}	T_{27}	T_{28}	T_{29}	T_{30}	T_{31}	T_{32}	T_{amb}
-	K	K	K	K	K	K	K	K	K	K	K
1	318.4	310.2	310.1	310.3	292.6	312.3	318.5	309.9	310.2	309.6	294.8
2	321.8	307.4	307.3	307.3	292.5	312.0	322.1	308.7	307.4	307.2	295.2
3	328.7	303.5	303.5	303.4	292.2	312.3	328.9	306.0	303.5	303.5	295.5
4	327.1	316.5	316.3	316.8	293.7	318.9	327.3	316.0	316.5	315.6	295.7
5	332.1	313.8	313.5	313.9	294.5	320.4	332.5	314.1	313.8	313.5	295.9
6	339.4	308.8	308.8	308.8	294.5	321.1	339.6	309.2	308.9	308.8	296.0
7	335.2	322.9	322.6	323.4	295.9	325.8	335.4	322.7	322.8	322.2	296.2
8	340.4	319.6	319.3	319.9	296.8	327.5	340.7	318.4	319.6	319.4	296.5
9	347.5	314.6	314.4	314.9	297.3	327.8	347.8	313.2	314.7	314.6	296.7
10	320.3	311.8	311.7	311.9	294.3	313.8	320.4	311.5	311.8	311.2	297.6
11	324.2	309.1	309.0	309.2	293.9	313.8	324.2	310.6	309.2	308.8	297.2
12	337.5	302.7	302.6	302.9	293.8	313.5	337.5	305.4	302.8	302.5	297.3
13	337.1	325.0	324.9	325.3	299.9	328.1	337.3	324.9	325.0	324.6	302.8
14	338.3	318.4	318.1	318.6	298.1	325.6	338.4	318.7	318.5	318.0	300.8
15	343.8	310.9	310.6	311.2	296.1	324.1	343.9	311.5	311.1	310.7	298.5
16	340.6	327.1	327.0	327.4	299.0	330.3	340.7	327.1	327.1	326.7	305.7
17	346.3	324.2	324.0	324.6	301.0	333.2	346.6	323.7	324.3	324.0	305.1
18	351.9	317.4	317.2	317.9	300.7	331.7	352.3	316.4	317.6	317.4	304.4
19	332.3	323.3	323.1	323.4	303.4	326.0	332.6	323.1	323.2	322.4	295.9
20	336.8	321.1	320.8	321.2	304.0	327.4	337.2	322.3	321.1	320.6	296.5
21	343.6	316.2	315.9	316.4	304.2	327.3	343.9	318.7	316.3	316.1	296.7
22	339.7	328.7	328.5	329.0	305.3	332.0	340.0	328.5	328.7	328.0	296.9
23	344.9	326.1	325.8	326.2	305.8	333.5	345.2	326.3	326.1	325.7	297.3
24	352.1	320.4	320.0	320.7	305.9	332.9	352.4	321.3	320.5	320.3	297.4
25	347.1	334.2	333.9	334.7	306.5	338.5	347.5	334.7	334.2	333.2	297.4
26	352.9	331.1	330.7	331.4	307.1	340.1	353.1	330.4	331.1	330.7	297.6
27	360.4	325.5	325.0	326.0	307.4	338.1	360.6	323.9	325.7	325.5	297.8

Continued on next page

Table D.1: Continued.

Run	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}
-	kPa	kPa	kPa	kPa	kPa	kPa	kPa	kPa	kPa	kPa	kPa
1	329.8	652.6	647.6	652.1	643.6	633.9	368.7	664.4	665.5	297.1	299.3
2	317.3	656.8	653.1	657.7	650.6	644.4	345.9	666.3	672.1	276.8	279.1
3	310.4	658.7	655.3	660.1	653.6	649.6	330.6	666.9	675.4	265.1	267.7
4	510.2	959.2	952.4	960.9	950.2	935.2	559.7	975.3	984.0	457.8	461.1
5	503.6	959.5	954.1	962.2	953.8	942.7	537.5	982.7	988.8	441.5	445.7
6	495.6	957.3	952.2	960.7	952.6	944.0	519.5	983.0	987.8	428.0	432.7
7	678.4	1206.2	1203.9	1208.2	1200.2	1182.8	729.7	1237.3	1243.3	606.7	611.6
8	676.4	1203.3	1201.4	1207.4	1200.5	1186.4	711.3	1242.2	1246.3	593.3	599.0
9	669.2	1200.4	1199.4	1205.0	1199.9	1188.7	694.5	1243.1	1245.5	581.9	588.4
10	316.8	690.7	684.6	683.3	681.9	670.8	346.2	698.5	689.9	284.5	284.6
11	308.3	694.3	690.5	688.6	687.3	679.7	330.7	694.8	698.2	268.0	269.3
12	294.0	699.8	695.7	695.0	693.9	687.7	308.3	697.4	703.9	248.9	250.5
13	580.8	1129.8	1120.4	1124.4	1123.5	1107.5	614.4	1139.3	1144.6	520.1	521.7
14	544.6	1098.2	1090.3	1092.8	1094.4	1082.3	567.6	1110.6	1115.7	477.4	480.0
15	520.8	1087.4	1080.1	1084.0	1083.3	1075.4	536.5	1100.2	1105.8	450.4	453.9
16	689.6	1335.1	1326.1	1325.3	1324.6	1308.6	727.4	1346.8	1351.4	616.3	619.3
17	690.1	1315.2	1308.2	1310.8	1310.0	1297.8	709.8	1336.9	1341.7	605.7	608.5
18	639.6	1271.0	1264.2	1267.9	1266.2	1259.0	651.9	1291.9	1295.8	558.3	561.5
19	378.3	834.5	825.6	834.4	824.8	814.6	411.9	838.6	846.8	339.7	341.1
20	375.1	837.1	829.3	840.2	829.3	822.9	398.5	846.8	853.3	328.2	330.0
21	368.3	836.8	829.9	840.7	830.7	826.2	383.2	849.5	855.3	316.8	319.1
22	520.1	1094.7	1091.3	1098.8	1088.4	1075.9	556.2	1109.9	1116.4	466.8	469.1
23	518.2	1094.8	1091.9	1099.6	1091.3	1081.6	542.5	1115.0	1120.6	455.4	458.5
24	511.4	1091.6	1089.1	1098.2	1089.2	1081.9	525.6	1115.3	1119.2	443.3	447.1
25	685.1	1392.4	1391.8	1396.4	1387.7	1373.9	725.0	1417.6	1424.1	613.8	617.5
26	686.2	1387.3	1387.7	1392.3	1385.2	1374.7	711.9	1421.0	1424.2	604.3	609.5
27	680.8	1377.7	1378.0	1384.4	1376.9	1370.1	694.8	1421.4	1416.2	594.3	599.6

Continued on next page

Table D.1: Continued.

Run	p_{12}	p_{13}	p_{14}	p_{16}	p_{17}	p_{19}	p_{20}	p_{21}	p_{22}	p_{23}	p_{24}
-	kPa	kPa	kPa	kPa	kPa	kPa	kPa	kPa	kPa	kPa	kPa
1	334.4	329.1	335.9	350.5	343.0	296.8	292.8	294.0	236.7	719.1	674.1
2	319.7	314.0	319.4	332.1	325.2	275.6	271.2	274.6	234.9	714.1	688.6
3	311.8	305.9	311.4	321.1	315.0	264.2	258.9	263.3	235.0	711.3	694.8
4	514.6	507.7	513.4	534.6	525.2	455.6	450.2	451.0	385.3	1054.5	1004.1
5	506.0	498.8	504.1	520.7	513.8	439.3	432.8	436.0	390.6	1049.0	1018.1
6	497.1	489.7	493.9	508.3	502.0	425.9	418.9	422.7	390.0	1042.3	1020.1
7	682.7	675.1	682.7	704.1	693.4	603.5	596.2	595.4	524.0	1330.3	1276.6
8	678.5	670.5	677.7	695.0	685.7	590.5	581.5	583.8	534.3	1323.5	1287.8
9	671.7	663.2	669.6	684.0	676.9	578.9	569.9	573.6	537.1	1317.2	1291.8
10	319.2	314.4	321.4	334.4	326.0	283.9	277.9	278.8	224.1	739.1	700.9
11	309.1	304.1	309.2	320.5	313.3	267.0	262.3	263.5	225.4	737.3	716.7
12	294.4	289.4	294.7	302.8	296.7	249.1	242.4	245.7	223.4	735.0	725.7
13	582.1	576.8	581.9	601.5	593.5	517.5	507.8	508.8	441.4	1218.5	1174.3
14	544.7	538.8	543.4	558.9	551.4	476.0	466.6	468.9	421.8	1177.4	1150.5
15	521.0	514.6	518.4	531.4	524.8	450.4	440.1	443.4	409.6	1160.8	1142.3
16	690.5	683.6	691.6	711.6	701.3	614.5	603.1	604.8	531.1	1436.0	1389.0
17	689.7	681.7	688.0	704.6	696.0	603.0	590.4	595.3	544.1	1417.3	1387.2
18	639.3	632.6	637.2	650.3	644.7	557.0	542.3	549.2	512.4	1361.2	1340.7
19	382.4	374.8	379.8	396.5	389.4	337.8	332.5	333.6	277.5	895.7	856.0
20	377.7	369.8	374.4	387.4	381.8	326.8	319.9	323.3	284.5	893.6	870.6
21	370.4	362.8	366.6	377.2	371.5	316.2	308.1	312.8	285.4	891.0	875.5
22	523.5	514.9	521.5	540.4	529.4	464.9	456.9	457.9	395.6	1177.5	1134.4
23	520.1	511.5	517.2	532.0	522.5	453.9	444.5	448.5	405.3	1174.0	1147.6
24	512.9	504.5	508.8	520.3	513.9	441.8	432.2	437.4	406.5	1167.4	1148.7
25	687.9	679.0	689.2	707.4	696.7	610.8	601.5	602.7	534.2	1496.9	1450.8
26	687.7	678.4	687.7	700.9	692.8	602.1	591.6	595.6	547.1	1491.0	1460.3
27	682.1	672.7	681.6	691.6	685.3	592.2	581.2	586.4	550.5	1478.8	1456.6

Continued on next page

Table D.1: Continued.

Run	p_{25}	p_{26}	p_{29}	p_{30}	p_{31}	p_{32}
-	kPa	kPa	kPa	kPa	kPa	kPa
1	320.7	686.6	703.6	693.9	682.3	680.3
2	288.2	692.9	703.6	698.2	691.1	688.2
3	271.8	695.9	704.0	700.2	695.3	691.9
4	479.5	1018.0	1037.9	1023.0	1011.9	1008.7
5	451.1	1023.7	1039.1	1028.0	1019.8	1015.8
6	433.3	1023.2	1036.2	1026.8	1020.4	1015.8
7	625.5	1288.7	1313.3	1297.7	1282.5	1278.0
8	600.7	1293.3	1312.8	1299.0	1289.7	1284.4
9	585.8	1293.3	1312.4	1298.4	1290.8	1285.1
10	305.9	710.6	735.0	715.8	707.1	703.2
11	276.7	719.0	736.2	721.5	717.0	714.5
12	252.7	723.9	738.2	725.6	723.9	719.6
13	537.0	1183.7	1214.8	1190.5	1180.3	1172.5
14	483.8	1153.4	1177.3	1157.3	1151.6	1144.6
15	453.7	1142.6	1162.3	1145.1	1141.2	1135.0
16	632.7	1397.7	1434.1	1406.8	1395.4	1384.0
17	609.4	1389.7	1420.7	1395.4	1388.8	1378.1
18	559.5	1341.2	1365.5	1345.0	1340.8	1331.4
19	362.8	867.2	884.0	873.9	863.0	860.4
20	338.9	874.4	886.0	879.5	872.4	868.9
21	323.6	876.5	885.6	880.1	875.4	871.3
22	487.0	1146.3	1165.2	1152.5	1141.0	1136.9
23	463.6	1152.0	1167.3	1156.0	1148.7	1143.8
24	448.1	1150.7	1164.1	1153.9	1148.4	1143.2
25	632.3	1462.1	1485.6	1469.9	1456.6	1452.0
26	611.3	1464.6	1483.0	1470.3	1460.9	1455.3
27	597.8	1457.9	1476.7	1462.7	1455.5	1449.9

Continued on next page

Table D.1: Continued.

Run	$\dot{m}_g$	$\dot{m}_{gly,h}$	$\dot{m}_{gly,c}$	τ_{comp}	τ_{exp}	τ_{motor}	τ_{pump}	N_{comp}	N_{exp}	N_{motor}	N_{pump}
-	kg/s	kg/s	kg/s	N-m	N-m	N-m	N-m	rpm	rpm	rpm	rpm
1	0.0142	0.04608	0.03944	7.050	2.046	0.006	3.417	3500	1736	476	236
2	0.0150	0.04593	0.03918	6.612	2.703	-0.134	3.562	3500	1736	238	117
3	0.0155	0.04598	0.03907	6.393	3.142	-0.142	3.701	3500	1734	115	54
4	0.0230	0.04623	0.03886	9.795	3.217	-0.421	3.751	3500	1736	476	236
5	0.0243	0.04613	0.03865	9.287	3.838	-0.540	3.873	3500	1735	239	117
6	0.0250	0.04605	0.03890	8.981	4.277	-0.556	4.103	3500	1733	116	53
7	0.0310	0.04574	0.03930	11.965	3.936	-0.756	4.002	3500	1734	476	236
8	0.0327	0.04558	0.03923	11.440	4.459	-0.836	4.082	3500	1733	239	117
9	0.0336	0.04560	0.03913	11.061	4.906	-0.872	4.278	3500	1734	117	54
10	0.0132	0.04831	0.01909	7.248	3.177	-0.089	3.534	3500	1155	476	236
11	0.0143	0.04646	0.02041	6.870	3.684	-0.219	3.688	3500	1156	239	116
12	0.0147	0.04872	0.01374	6.587	4.128	-0.219	3.883	3500	1156	74	55
13	0.0257	0.04688	0.02428	11.062	4.826	-0.709	4.052	3500	1155	475	236
14	0.0259	0.04772	0.01859	10.351	5.399	-0.802	4.155	3500	1155	239	117
15	0.0261	0.04801	0.01726	10.008	5.869	-0.797	4.390	3500	1155	118	55
16	0.0308	0.04879	0.01667	12.810	5.788	-0.980	4.379	3500	1160	475	236
17	0.0327	0.04682	0.01589	12.217	6.153	-1.077	4.459	3500	1156	238	117
18	0.0318	0.04780	0.00704	11.527	6.626	-1.042	4.671	3500	1155	118	55
19	0.0158	0.04288	0.05710	8.341	4.055	-0.353	3.841	3500	863	476	236
20	0.0171	0.04381	0.04754	7.935	4.564	-0.439	3.993	3500	863	239	117
21	0.0177	0.04381	0.04760	7.712	4.962	-0.466	4.080	3500	863	116	54
22	0.0223	0.04630	0.04973	10.579	5.258	-0.720	4.142	3500	864	476	236
23	0.0239	0.04612	0.04967	10.146	5.821	-0.827	4.289	3500	864	239	117
24	0.0247	0.04609	0.04966	9.894	6.269	-0.847	4.405	3500	864	117	54
25	0.0299	0.04584	0.04988	13.144	6.624	-1.172	4.588	3500	864	476	236
26	0.0318	0.04561	0.04986	12.647	7.201	-1.282	4.718	3500	863	239	117
27	0.0328	0.04623	0.04163	12.285	7.677	-1.252	4.850	3500	863	118	54

Continued on next page

Table D.1: Continued.

Run	$x_{l,h}$	$x_{l,c}$	$\dot{m}_{l,h}$	$\dot{m}_{l,c}$	$\eta_{i,comp}$	$\eta_{i,exp}$	$\eta_{v,comp}$	$\eta_{v,exp}$
-	-	-	kg/s	kg/s	-	-	-	-
1	0.911	0.766	0.1454	0.0466	0.599	0.554	0.932	1.049
2	0.828	0.622	0.0725	0.0248	0.655	0.618	0.967	1.074
3	0.706	0.431	0.0373	0.0117	0.687	0.656	0.977	1.092
4	0.861	0.672	0.1418	0.0470	0.642	0.574	0.942	1.143
5	0.748	0.482	0.0724	0.0226	0.687	0.604	0.960	1.189
6	0.601	0.275	0.0377	0.0095	0.715	0.631	0.963	1.214
7	0.818	0.593	0.1398	0.0451	0.668	0.553	0.951	1.223
8	0.690	0.383	0.0727	0.0203	0.701	0.573	0.959	1.277
9	0.533	0.182	0.0384	0.0075	0.727	0.597	0.957	1.305
10	0.916	0.778	0.1451	0.0465	0.588	0.503	0.922	1.394
11	0.837	0.627	0.0735	0.0240	0.645	0.508	0.966	1.464
12	0.639	0.419	0.0261	0.0106	0.685	0.513	0.972	1.474
13	0.843	0.618	0.1382	0.0416	0.659	0.441	0.943	1.652
14	0.740	0.437	0.0736	0.0201	0.693	0.464	0.960	1.667
15	0.596	0.223	0.0384	0.0075	0.716	0.482	0.962	1.668
16	0.819	0.600	0.1397	0.0463	0.674	0.447	0.946	1.664
17	0.691	0.343	0.0730	0.0170	0.704	0.447	0.955	1.766
18	0.547	0.149	0.0385	0.0056	0.726	0.475	0.956	1.734
19	0.899	0.735	0.1414	0.0438	0.620	0.380	0.920	1.885
20	0.812	0.539	0.0738	0.0200	0.668	0.382	0.947	1.992
21	0.683	0.289	0.0382	0.0072	0.697	0.392	0.954	2.040
22	0.864	0.644	0.1419	0.0404	0.653	0.362	0.926	2.021
23	0.757	0.419	0.0746	0.0172	0.691	0.368	0.944	2.127
24	0.612	0.157	0.0390	0.0046	0.714	0.382	0.946	2.178
25	0.825	0.565	0.1412	0.0388	0.675	0.353	0.932	2.119
26	0.703	0.316	0.0753	0.0147	0.707	0.360	0.943	2.234
27	0.550	0.059	0.0402	0.0021	0.730	0.376	0.942	2.297

Table D.2: Additional Tests from LFEC Testing.

Run	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}
-	K	K	K	K	K	K	K	K	K	K	K
1	299.2	296.2	295.5	295.5	295.4	295.4	284.3	302.4	295.4	300.6	293.9
2	288.4	291.0	290.6	291.8	291.0	291.3	279.3	305.5	292.0	302.0	288.7
3	290.3	291.6	291.1	293.9	291.8	291.9	285.9	309.9	294.3	305.9	290.5
4	292.0	293.0	292.7	295.7	293.1	293.1	290.4	312.7	296.1	308.6	292.3
5	293.3	294.5	294.1	298.7	295.2	295.1	290.9	322.5	299.1	315.9	293.2
6	290.5	292.3	291.8	295.6	293.0	293.3	284.9	316.9	295.7	310.9	290.3
7	293.1	299.9	298.0	293.7	294.3	293.8	274.1	316.2	294.4	309.2	287.6
8	290.7	292.9	292.4	297.0	294.2	294.7	284.4	322.5	297.8	314.7	290.3
9	275.3	278.8	277.8	281.0	278.5	279.5	267.4	300.8	281.1	295.8	275.9
10	282.1	283.7	282.9	286.6	283.8	284.1	277.2	304.3	286.9	299.8	282.5
11	286.5	287.6	287.1	290.6	287.8	287.8	284.7	307.7	291.1	303.4	286.9
12	287.8	288.9	288.4	291.8	289.0	289.0	285.9	308.5	292.2	304.3	288.2
13	286.9	288.1	287.6	292.1	288.6	288.6	284.2	313.6	292.5	307.6	287.0
14	285.0	286.7	286.0	290.1	287.2	287.6	278.7	310.4	290.2	304.7	285.0
15	282.4	285.6	284.6	286.9	285.5	286.1	270.4	305.1	287.1	299.9	282.2
16	287.1	289.4	288.7	289.9	289.1	289.4	274.7	301.4	290.0	298.4	287.1
17	269.8	272.4	270.5	275.4	272.4	274.0	257.5	309.4	275.9	298.9	265.7
18	282.9	284.3	283.7	289.4	285.8	286.3	275.9	315.4	290.3	307.4	282.5
19	286.0	287.2	286.7	292.9	288.3	288.4	282.9	320.2	293.8	311.9	285.9
20	286.5	287.7	287.2	293.3	288.8	288.8	283.4	320.3	294.2	312.1	286.4
21	282.5	286.1	285.2	287.7	286.2	286.9	269.9	310.1	287.9	303.2	281.6
22	285.5	289.0	288.4	289.5	288.6	289.1	273.6	307.5	289.3	302.4	285.0
23	284.9	288.9	288.4	290.5	289.2	289.9	272.6	314.8	291.1	307.2	284.0
24	274.0	279.4	277.8	279.4	278.2	279.3	259.6	308.6	279.9	299.8	271.3
25	272.5	278.3	276.5	277.1	276.5	277.4	257.6	303.8	277.5	295.9	269.5
26	266.6	273.7	271.4	272.5	271.7	272.8	253.2	299.5	272.5	291.5	264.5
27	269.4	272.5	271.3	277.9	273.9	275.2	260.4	312.4	278.4	302.2	268.7

Continued on next page

Table D.2: Continued.

Run	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}
-	K	K	K	K	K	K	K	K	K	K	K
28	274.1	276.4	275.6	282.2	277.8	278.8	266.6	314.5	282.9	304.9	273.7
29	277.3	279.3	278.3	285.2	280.5	281.4	271.1	316.5	286.0	307.1	277.1
30	279.2	281.0	280.0	287.1	282.2	282.8	273.9	318.2	287.7	308.8	279.0
31	280.0	281.6	280.9	287.9	282.8	283.4	275.6	318.9	288.4	309.6	279.9
32	280.1	281.6	281.1	288.2	282.8	283.3	276.5	319.4	288.6	310.0	280.1
33	280.2	281.7	281.2	288.4	282.8	283.3	277.1	319.8	288.8	310.4	280.3
34	284.9	286.4	285.8	290.0	286.4	286.6	283.5	304.4	290.2	300.9	286.5
35	281.5	283.1	282.3	286.8	282.9	283.2	279.6	302.7	286.9	299.0	282.8
36	279.8	281.7	280.8	285.3	281.5	281.9	277.7	301.6	285.5	297.7	281.3
37	280.7	283.0	281.6	285.9	282.5	282.9	278.0	300.7	285.7	297.1	282.0

Continued on next page

Table D.2: Continued.

Run	T_{12}	T_{13}	T_{14}	T_{15}	T_{16}	T_{17}	T_{18}	T_{19}	T_{20}	T_{21}	T_{22}
-	K	K	K	K	K	K	K	K	K	K	K
1	298.6	293.6	292.0	295.6	284.0	291.8	290.0	302.9	300.8	302.0	302.5
2	288.2	288.0	287.5	296.2	278.4	287.6	284.5	306.2	302.5	304.7	305.4
3	290.0	290.0	289.8	296.7	285.4	289.7	289.1	311.2	306.4	310.1	310.8
4	291.8	291.9	291.8	297.2	290.0	291.6	291.9	314.4	309.1	313.4	314.2
5	293.1	293.1	293.1	299.9	290.5	292.9	293.5	324.1	316.7	322.3	323.5
6	290.1	290.0	289.7	297.3	284.3	289.6	289.2	318.0	311.7	315.9	317.0
7	289.2	287.3	286.9	298.5	273.5	287.0	288.0	316.4	310.2	312.9	314.3
8	290.2	290.1	290.0	299.0	283.8	290.2	289.8	323.3	315.7	320.2	321.8
9	274.8	274.5	273.3	292.1	265.2	273.3	268.7	301.3	296.7	299.4	300.4
10	281.7	281.7	281.5	291.3	276.1	281.2	280.8	305.2	300.5	304.0	304.7
11	286.3	286.4	286.3	292.0	284.2	286.0	286.9	309.0	304.0	308.0	308.9
12	287.6	287.6	287.5	292.4	285.4	287.3	288.3	309.7	304.8	308.7	309.5
13	286.7	286.7	286.6	293.8	283.7	286.4	287.6	314.9	308.4	313.3	314.5
14	284.6	284.6	284.3	293.7	277.7	284.0	284.6	311.3	305.4	309.3	310.4
15	282.1	281.7	281.3	293.1	268.6	281.3	281.1	305.5	300.6	303.2	304.2
16	286.8	286.6	286.3	291.0	273.1	286.3	287.9	301.8	298.8	300.5	301.2
17	266.1	264.7	263.6	294.4	255.3	264.4	259.0	309.7	300.6	305.1	306.7
18	282.4	282.3	282.2	294.5	274.9	282.0	283.0	316.2	308.4	312.9	314.6
19	285.9	285.8	285.8	294.9	282.5	285.4	287.3	321.5	312.9	319.0	320.6
20	286.3	286.3	286.3	295.1	283.0	285.8	287.7	321.5	313.1	319.2	320.6
21	281.8	281.2	280.8	294.5	268.6	281.8	281.2	310.4	304.1	307.0	308.3
22	285.0	284.6	284.3	293.6	272.8	284.5	283.7	308.1	303.1	305.7	306.8
23	284.3	283.8	283.7	296.2	271.9	284.6	284.1	315.0	308.2	311.5	312.8
24	272.3	270.5	290.1	289.8	257.7	270.8	265.5	308.8	301.1	304.4	306.1
25	270.5	268.5	289.9	290.6	255.4	268.8	262.7	304.2	297.1	299.2	300.7
26	265.8	263.3	288.9	291.5	250.5	263.2	255.7	300.0	292.9	294.6	295.7
27	268.7	268.2	289.3	291.3	258.4	269.0	263.9	313.2	303.7	308.8	310.7

Continued on next page

Table D.2: Continued.

Run	T_{12}	T_{13}	T_{14}	T_{15}	T_{16}	T_{17}	T_{18}	T_{19}	T_{20}	T_{21}	T_{22}
-	K	K	K	K	K	K	K	K	K	K	K
28	273.7	273.4	290.0	291.0	265.0	273.7	271.2	315.5	306.2	311.5	313.5
29	277.0	276.7	290.7	291.0	269.7	276.4	275.5	317.7	308.4	314.0	316.0
30	278.8	278.7	291.3	291.1	272.7	278.0	278.0	319.5	310.1	316.3	318.1
31	279.6	279.5	291.5	290.5	274.4	278.6	279.1	320.2	310.9	317.3	319.0
32	279.8	279.7	291.7	289.9	275.3	278.6	279.5	320.7	311.3	317.9	319.6
33	279.9	279.9	292.0	289.5	276.0	278.8	279.7	321.1	311.7	318.5	320.1
34	284.6	285.4	292.8	286.5	282.5	283.6	283.8	305.8	301.4	305.0	305.7
35	281.1	281.6	291.8	283.7	278.4	279.9	280.1	303.3	299.6	302.6	303.2
36	279.6	280.0	291.4	282.9	276.5	278.4	278.5	302.1	298.4	301.3	301.9
37	280.2	280.6	291.7	283.1	276.6	278.8	278.7	301.6	297.8	300.7	301.3

Continued on next page

Table D.2: Continued.

Run	T_{23}	T_{24}	T_{25}	T_{26}	T_{27}	T_{28}	T_{29}	T_{30}	T_{31}	T_{32}	T_{amb}
-	K	K	K	K	K	K	K	K	K	K	K
1	325.2	303.1	302.9	303.3	294.9	312.1	325.1	304.9	303.2	302.8	298.4
2	333.4	306.0	306.2	306.7	295.9	317.2	333.3	309.3	306.5	306.1	299.8
3	326.6	311.3	311.2	311.4	296.3	315.6	326.6	312.5	311.4	310.8	300.6
4	323.1	314.4	314.4	314.5	296.8	316.2	323.1	314.0	314.4	314.0	301.6
5	336.1	324.2	324.1	324.4	300.1	327.1	336.3	323.8	324.2	323.7	303.7
6	338.7	318.1	318.0	318.2	297.0	324.7	338.8	318.1	318.2	317.8	304.2
7	353.5	316.8	316.5	317.3	298.6	331.6	353.8	315.3	317.0	316.8	305.1
8	347.2	323.5	323.3	323.9	299.1	332.7	347.4	322.7	323.6	323.3	304.8
9	327.7	301.3	301.3	301.4	291.0	304.8	328.1	305.2	301.4	301.2	297.2
10	320.6	305.2	305.1	305.2	291.0	306.3	320.9	307.0	305.3	304.9	297.9
11	318.0	309.0	309.0	309.1	291.8	308.2	318.3	309.1	309.0	308.7	298.5
12	318.6	309.6	309.6	309.7	292.2	308.8	318.9	309.7	309.6	309.3	299.7
13	326.0	315.0	314.9	315.1	293.7	314.1	326.3	314.9	315.0	314.6	298.9
14	330.0	311.4	311.3	311.5	293.5	313.3	330.3	312.6	311.5	311.1	296.6
15	336.6	305.4	305.6	305.8	292.9	311.7	337.1	307.8	305.8	305.5	296.0
16	328.6	301.8	301.8	302.0	291.0	307.0	329.1	305.6	301.9	301.7	295.9
17	344.2	310.0	309.8	310.6	294.3	318.0	345.0	310.6	310.3	310.0	295.5
18	337.9	316.5	316.2	316.9	294.4	319.8	338.4	315.6	316.6	316.2	295.8
19	334.4	321.6	321.5	322.0	294.9	321.1	334.9	320.9	321.6	321.2	296.4
20	334.4	321.6	321.5	322.0	295.1	321.1	334.9	321.0	321.6	321.3	296.6
21	344.2	310.7	310.5	310.9	294.4	318.9	344.9	310.7	310.7	310.6	296.7
22	343.5	308.4	308.2	308.7	293.4	316.2	344.2	310.3	308.6	308.2	297.5
23	354.8	315.6	315.2	316.3	296.2	326.6	355.5	314.5	315.8	315.5	297.4
24	353.9	309.2	309.0	309.7	289.8	320.5	354.5	306.6	309.4	309.1	297.8
25	367.6	304.1	304.2	303.9	290.6	318.8	368.2	302.4	303.9	304.0	298.2
26	383.1	299.7	300.0	299.5	291.2	318.6	383.2	299.0	299.6	299.6	298.8
27	348.6	313.6	313.3	314.0	290.9	323.4	348.8	310.3	313.7	313.1	299.2

Continued on next page

Table D.2: Continued.

Run	T_{23}	T_{24}	T_{25}	T_{26}	T_{27}	T_{28}	T_{29}	T_{30}	T_{31}	T_{32}	T_{amb}
-	K	K	K	K	K	K	K	K	K	K	K
28	344.2	315.8	315.5	316.0	290.8	324.8	344.3	313.8	315.8	315.3	299.2
29	341.8	317.9	317.6	318.1	290.8	324.7	341.9	316.3	318.0	317.4	299.4
30	340.6	319.8	319.6	320.0	291.0	324.5	340.7	318.0	319.7	319.2	299.5
31	339.0	320.4	320.2	320.6	290.2	323.8	339.1	318.9	320.4	319.9	299.6
32	337.7	320.8	320.7	321.0	289.6	323.4	337.8	319.8	320.8	320.3	299.8
33	336.8	321.2	321.1	321.4	289.4	323.5	337.0	320.6	321.2	320.7	299.9
34	315.6	305.7	305.7	305.8	285.6	307.5	315.6	305.3	305.7	305.2	295.5
35	314.3	303.2	303.3	303.1	283.0	304.7	314.3	302.7	303.2	303.1	296.1
36	314.3	302.0	302.0	301.9	282.2	303.6	314.4	301.7	301.9	301.9	296.3
37	315.6	301.5	301.5	301.4	282.4	303.5	315.6	301.5	301.5	301.2	296.5

Continued on next page

Table D.2: Continued.

Run	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}
-	kPa	kPa	kPa	kPa	kPa	kPa	kPa	kPa	kPa	kPa	kPa
1	324.8	526.3	520.7	523.0	521.8	519.5	321.7	561.0	543.0	272.7	275.7
2	296.5	741.9	728.6	733.6	730.5	725.8	311.3	737.5	740.4	253.3	255.0
3	304.9	734.2	720.0	723.7	722.6	715.0	325.4	734.3	730.8	265.4	266.1
4	316.0	731.2	715.7	719.4	717.9	707.0	343.7	728.8	724.3	283.9	283.7
5	527.6	1128.5	1118.9	1118.0	1116.2	1101.8	563.5	1127.8	1132.3	473.0	474.3
6	519.3	1136.6	1128.1	1127.0	1127.9	1115.7	544.1	1139.8	1144.1	454.8	457.1
7	691.8	1401.7	1393.8	1392.6	1394.7	1387.7	694.7	1439.6	1425.4	603.1	607.2
8	699.2	1446.0	1436.6	1435.5	1438.5	1425.9	721.5	1458.8	1466.7	613.8	617.2
9	313.3	688.4	682.9	683.0	680.9	672.7	336.8	690.8	693.8	268.4	269.6
10	329.5	685.6	678.6	679.3	677.2	666.5	357.6	693.5	690.2	287.3	287.7
11	355.4	693.5	685.1	685.2	683.2	668.1	390.1	695.2	695.7	319.1	318.6
12	358.0	694.5	685.9	686.3	684.3	669.4	391.7	698.0	696.7	321.1	320.5
13	496.9	925.4	916.6	918.1	913.0	902.3	537.7	928.5	936.3	445.0	445.7
14	486.1	929.0	921.4	923.4	919.3	912.3	514.7	936.8	943.9	424.9	426.7
15	481.3	944.7	937.6	939.9	936.7	932.0	502.1	956.2	962.1	414.8	417.6
16	339.8	705.7	700.7	700.7	698.2	692.3	358.9	711.1	714.5	290.0	292.0
17	619.3	1237.5	1234.4	1233.7	1235.8	1225.9	658.1	1258.0	1262.3	540.3	545.2
18	683.2	1230.9	1226.6	1226.5	1228.6	1215.4	715.6	1269.7	1261.8	598.1	601.3
19	689.5	1217.7	1211.2	1211.0	1212.6	1194.9	735.1	1236.2	1244.0	614.5	616.5
20	680.1	1199.6	1193.0	1193.3	1195.0	1177.4	725.0	1240.5	1225.7	606.4	608.4
21	653.6	1197.2	1192.0	1192.9	1195.3	1185.6	675.7	1221.8	1227.8	566.9	570.7
22	476.2	1137.0	1130.0	1131.5	1132.6	1126.5	492.5	1151.8	1148.7	410.8	413.0
23	686.3	1510.2	1507.9	1507.2	1508.0	1499.8	703.8	1534.2	1535.0	597.6	601.6
24	610.9	1491.0	1492.5	1492.6	1490.3	1483.0	638.2	1526.5	1515.0	533.0	535.6
25	593.4	1494.7	1495.9	1497.0	1493.9	1487.0	623.9	1526.4	1516.5	516.6	518.8
26	565.0	1507.0	1508.0	1508.5	1506.3	1499.0	603.2	1528.5	1524.6	494.0	495.4
27	604.9	1509.5	1509.9	1510.4	1508.3	1499.4	641.3	1530.9	1529.6	531.9	533.5

Continued on next page

Table D.2: Continued.

Run	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}
-	kPa	kPa	kPa	kPa	kPa	kPa	kPa	kPa	kPa	kPa	kPa
28	629.7	1503.6	1503.6	1503.7	1501.9	1491.9	662.6	1531.2	1525.2	555.0	556.4
29	647.5	1502.7	1502.8	1502.2	1500.8	1489.8	680.2	1532.4	1525.5	572.5	574.0
30	660.2	1507.3	1506.9	1507.8	1505.0	1493.1	694.4	1533.1	1530.3	586.2	587.5
31	666.9	1509.4	1508.8	1508.3	1506.2	1493.7	704.2	1533.6	1531.6	594.2	595.2
32	670.8	1511.9	1510.9	1512.5	1508.2	1495.0	711.0	1534.0	1533.2	600.1	600.8
33	673.7	1515.3	1513.6	1511.7	1510.7	1496.8	717.3	1535.3	1535.8	604.8	605.6
34	314.1	768.3	758.4	763.4	758.2	748.4	350.1	758.9	769.9	282.0	282.6
35	301.6	771.6	762.3	766.4	762.5	753.2	337.6	764.0	773.0	269.9	270.3
36	295.2	773.9	765.3	768.4	765.5	756.8	329.7	766.8	775.9	262.6	263.0
37	298.1	783.2	774.6	779.3	774.6	767.1	330.7	767.2	786.5	262.8	263.4

Continued on next page

Table D.2: Continued.

Run	p_{12}	p_{13}	p_{14}	p_{16}	p_{17}	p_{19}	p_{20}	p_{21}	p_{22}	p_{23}	p_{24}
-	kPa	kPa	kPa	kPa	kPa	kPa	kPa	kPa	kPa	kPa	kPa
1	325.3	316.7	319.7	324.5	319.7	274.1	266.7	269.1	240.4	588.7	570.9
2	297.2	291.9	297.0	305.0	299.4	255.5	246.8	249.8	222.6	774.4	761.0
3	305.5	299.7	306.3	316.2	309.0	267.3	258.1	260.8	221.9	768.9	748.4
4	318.2	312.9	319.6	332.3	324.3	284.8	276.7	278.0	222.3	772.2	735.3
5	529.2	522.8	527.7	547.6	537.8	472.5	462.3	463.0	395.0	1199.0	1157.5
6	519.0	512.8	516.6	533.0	526.0	455.1	443.3	447.4	399.7	1201.8	1177.0
7	690.6	682.4	688.6	698.1	691.1	600.9	588.5	594.2	553.0	1495.1	1473.6
8	698.1	691.0	697.9	714.1	704.7	611.7	599.3	604.0	549.2	1540.2	1510.3
9	313.9	310.2	316.0	325.4	319.7	268.7	261.7	263.7	236.8	731.3	716.8
10	330.7	326.7	333.4	344.4	337.9	288.1	279.6	281.6	243.0	735.0	711.6
11	358.3	353.4	360.7	376.1	367.4	318.2	311.1	311.8	254.2	754.3	711.1
12	360.6	355.8	362.9	378.0	369.7	320.1	312.7	313.8	256.1	755.0	712.2
13	499.3	493.6	498.2	520.3	508.1	443.5	434.4	435.1	368.7	1008.1	961.2
14	486.7	480.6	485.0	502.5	494.6	423.3	414.0	417.0	369.6	1004.3	977.1
15	481.6	475.3	478.7	493.5	487.0	413.3	404.0	408.2	372.2	1016.5	997.7
16	340.5	335.8	342.1	349.8	344.8	289.3	282.8	284.7	256.7	755.1	738.1
17	620.5	614.8	620.5	639.1	630.1	539.3	529.3	529.7	493.5	1330.2	1304.6
18	683.7	677.9	683.6	702.1	693.2	595.7	584.8	586.0	533.5	1342.8	1306.8
19	691.6	685.8	692.2	715.6	704.2	611.7	601.4	600.6	527.0	1334.7	1280.5
20	681.8	676.3	682.8	706.1	695.1	603.9	593.4	593.0	519.7	1315.7	1260.9
21	653.5	647.2	652.3	667.1	660.1	565.3	553.0	556.4	518.0	1299.9	1273.5
22	475.7	469.1	473.6	486.2	478.9	410.5	400.0	404.0	369.5	1196.8	1176.8
23	685.6	677.9	683.3	697.8	689.4	595.3	583.6	587.3	547.2	1599.8	1576.6
24	612.0	603.2	610.4	624.9	616.8	531.9	517.7	522.9	490.4	1567.0	1548.0
25	594.9	586.0	593.8	608.1	600.3	515.9	501.1	506.8	481.9	1563.2	1549.5
26	566.7	558.2	567.0	583.8	574.0	493.7	478.6	485.3	464.9	1564.8	1554.1
27	607.0	598.2	606.5	624.9	614.5	529.9	515.9	522.0	483.1	1583.1	1560.8

Continued on next page

Table D.2: Continued.

Run	p_{12}	p_{13}	p_{14}	p_{16}	p_{17}	p_{19}	p_{20}	p_{21}	p_{22}	p_{23}	p_{24}
-	kPa	kPa	kPa	kPa	kPa	kPa	kPa	kPa	kPa	kPa	kPa
28	631.7	622.9	630.9	649.6	639.9	552.4	538.5	544.4	499.0	1583.0	1557.3
29	649.4	641.1	648.8	668.7	658.1	569.7	556.1	561.5	510.4	1586.6	1557.5
30	662.9	654.2	662.0	682.0	672.5	583.0	569.7	574.4	518.0	1594.1	1562.0
31	669.6	661.3	669.4	690.9	680.2	591.0	577.7	582.0	520.7	1598.2	1561.0
32	673.9	665.9	673.9	696.2	685.9	596.9	583.5	587.3	521.7	1601.8	1562.1
33	677.6	669.6	677.7	701.3	689.2	601.6	588.3	591.9	522.1	1606.1	1563.6
34	318.9	312.1	320.3	334.3	324.9	284.0	274.3	276.6	224.7	813.4	775.8
35	306.3	300.1	308.3	322.1	312.6	271.4	262.4	264.7	217.3	813.8	779.1
36	299.5	293.6	302.2	314.9	305.4	263.7	254.8	257.5	214.0	813.6	783.5
37	301.5	295.3	303.7	316.0	307.2	263.6	254.9	258.3	217.7	821.9	796.3

Continued on next page

Table D.2: Continued.

Run	p_{25}	p_{26}	p_{29}	p_{30}	p_{31}	p_{32}
-	kPa	kPa	kPa	kPa	kPa	kPa
1	277.5	570.7	590.0	575.1	570.5	566.6
2	259.4	759.9	774.4	762.9	760.0	755.5
3	275.9	750.2	768.6	753.6	749.2	744.5
4	306.0	744.2	769.5	749.8	741.3	736.7
5	491.4	1167.0	1194.2	1172.2	1162.6	1155.8
6	462.0	1179.3	1203.5	1182.9	1177.9	1168.8
7	604.8	1473.2	1501.4	1478.2	1473.4	1462.7
8	618.7	1512.4	1542.9	1518.7	1512.2	1501.1
9	274.9	716.5	731.9	718.7	715.5	713.1
10	298.6	714.0	733.4	717.2	712.4	709.1
11	342.6	721.0	748.1	726.8	717.4	713.6
12	344.0	722.3	749.9	728.4	719.0	714.4
13	466.5	972.3	1001.2	978.8	968.0	962.4
14	434.1	980.6	1002.2	984.7	978.5	972.0
15	419.9	998.8	1018.9	1002.2	998.2	991.1
16	295.9	739.5	755.9	742.4	738.6	735.6
17	544.6	1308.2	1330.4	1310.5	1304.3	1301.5
18	605.1	1312.8	1344.0	1316.8	1308.8	1303.8
19	632.5	1294.3	1332.8	1300.9	1288.0	1282.5
20	624.4	1275.5	1313.6	1281.9	1269.4	1263.5
21	570.1	1278.0	1306.3	1280.9	1275.7	1268.6
22	416.8	1179.6	1198.3	1182.6	1179.1	1173.2
23	601.6	1578.4	1605.9	1583.6	1578.5	1571.1
24	537.8	1547.7	1567.9	1552.5	1545.3	1541.4
25	520.3	1547.4	1566.8	1551.3	1545.3	1541.4
26	498.6	1551.5	1570.3	1555.3	1549.9	1545.8
27	537.9	1561.3	1583.8	1565.5	1557.6	1553.0

Continued on next page

Table D.2: Continued.

Run	p_{25}	p_{26}	p_{29}	p_{30}	p_{31}	p_{32}
-	kPa	kPa	kPa	kPa	kPa	kPa
28	562.5	1558.5	1582.6	1563.4	1554.5	1549.9
29	582.0	1559.6	1585.2	1564.8	1555.1	1550.4
30	597.6	1565.0	1592.1	1570.6	1560.1	1555.6
31	608.3	1567.0	1595.3	1572.9	1561.2	1557.1
32	617.3	1568.2	1598.3	1575.1	1562.4	1558.4
33	625.0	1570.6	1601.8	1577.7	1564.4	1560.1
34	309.0	785.3	807.6	793.5	780.7	780.1
35	295.5	787.6	808.3	795.0	783.4	782.4
36	284.9	789.9	808.9	797.0	786.6	784.9
37	282.1	800.3	817.9	807.1	797.9	796.0

Continued on next page

Table D.2: Continued.

Run	$\dot{m}_g$	$\dot{m}_{gly,h}$	$\dot{m}_{gly,c}$	τ_{comp}	τ_{exp}	τ_{motor}	τ_{pump}	N_{comp}	N_{exp}	N_{motor}	N_{pump}
-	kg/s	kg/s	kg/s	N-m	N-m	N-m	N-m	rpm	rpm	rpm	rpm
1	0.0160	0.04929	0.00548	5.518	1.931	0.246	3.510	3500	863	115	54
2	0.0145	0.05079	0.00796	6.943	4.615	-0.298	4.022	3500	863	115	54
3	0.0140	0.05043	0.01132	7.082	4.166	-0.252	3.848	3500	864	239	117
4	0.0130	0.05032	0.01145	7.471	3.667	-0.106	3.773	3500	864	476	236
5	0.0230	0.04941	0.01479	10.900	5.557	-0.756	4.288	3500	863	475	236
6	0.0245	0.04961	0.01414	10.529	6.242	-0.908	4.451	3500	864	239	117
7	0.0344	0.04933	0.01339	12.569	7.969	-1.271	4.949	3500	865	117	54
8	0.0331	0.04915	0.01460	13.183	7.752	-1.394	4.879	3500	865	238	117
9	0.0160	0.05083	0.00421	6.686	3.137	-0.190	3.630	3500	1751	116	55
10	0.0159	0.05055	0.01097	6.904	2.799	-0.187	3.421	3500	1751	239	117
11	0.0156	0.05014	0.01513	7.456	2.207	-0.040	3.286	3500	1751	476	236
12	0.0156	0.04976	0.01805	7.460	2.216	-0.055	3.291	3500	1751	476	236
13	0.0225	0.04961	0.01761	9.514	3.051	-0.349	3.540	3500	1750	476	236
14	0.0237	0.04986	0.01635	9.061	3.721	-0.485	3.669	3500	1751	239	117
15	0.0246	0.05007	0.01433	8.909	4.261	-0.552	3.903	3500	1751	116	55
16	0.0172	0.04725	0.04093	6.864	3.335	-0.258	3.610	3500	1751	116	54
17	0.0319	0.04882	0.00377	11.427	5.446	-1.090	4.226	3500	1751	117	54
18	0.0335	0.04768	0.01617	11.772	4.680	-0.956	3.936	3500	1751	239	117
19	0.0317	0.04740	0.01787	12.185	3.925	-0.802	3.792	3500	1751	476	236
20	0.0312	0.04750	0.01803	12.036	3.858	-0.762	3.743	3500	1751	476	236
21	0.0333	0.04769	0.01466	11.089	5.044	-0.914	4.169	3500	1751	118	54
22	0.0240	0.04944	0.01411	10.366	7.172	-1.084	4.652	3500	863	117	55
23	0.0344	0.04859	0.01414	13.581	9.176	-1.654	5.223	3500	864	117	55
24	0.0310	0.04827	0.00707	13.287	9.872	-1.768	5.208	3500	892	95	39
25	0.0304	0.05009	0.00604	13.226	10.123	-1.689	5.223	3500	892	42	39
26	0.0288	0.05017	0.00430	13.234	10.515	-1.745	5.248	3500	892	22	39
27	0.0300	0.05003	0.00675	13.495	9.858	-1.831	5.111	3500	893	140	59

Continued on next page

Table D.2: Continued.

Run	$\dot{m}_g$	$\dot{m}_{gly,h}$	$\dot{m}_{gly,c}$	τ_{comp}	τ_{exp}	τ_{motor}	τ_{pump}	N_{comp}	N_{exp}	N_{motor}	N_{pump}
-	kg/s	kg/s	kg/s	N-m	N-m	N-m	N-m	rpm	rpm	rpm	rpm
28	0.0307	0.04975	0.00979	13.547	9.396	-1.764	4.978	3500	892	199	93
29	0.0310	0.04959	0.01173	13.636	9.044	-1.702	4.814	3500	892	239	118
30	0.0312	0.04942	0.01281	13.763	8.795	-1.665	4.781	3500	891	298	137
31	0.0311	0.04952	0.01336	13.896	8.637	-1.626	4.728	3500	892	339	158
32	0.0308	0.04951	0.01364	14.036	8.482	-1.582	4.729	3500	893	397	197
33	0.0306	0.04956	0.01371	14.157	8.408	-1.554	4.732	3500	892	444	216
34	0.0134	0.04856	0.01507	7.802	4.170	-0.273	3.590	3500	892	444	216
35	0.0131	0.04901	0.01367	7.722	4.384	-0.325	3.644	3500	892	397	197
36	0.0132	0.04914	0.01299	7.651	4.545	-0.379	3.700	3500	891	339	159
37	0.0136	0.04920	0.01283	7.660	4.712	-0.403	3.817	3500	892	298	138

Continued on next page

Table D.2: Continued.

Run	$x_{l,h}$	$x_{l,c}$	$\dot{m}_{l,h}$	$\dot{m}_{l,c}$	$\eta_{i,comp}$	$\eta_{i,exp}$	$\eta_{v,comp}$	$\eta_{v,exp}$
-	-	-	kg/s	kg/s	-	-	-	-
1	0.697	-0.003	0.0369	-0.0000	0.661	0.277	0.987	2.872
2	0.724	0.416	0.0381	0.0104	0.675	0.412	0.976	1.851
3	0.843	0.645	0.0749	0.0253	0.645	0.402	0.963	1.824
4	0.919	0.806	0.1477	0.0540	0.593	0.399	0.923	1.769
5	0.859	0.678	0.1401	0.0484	0.655	0.371	0.941	1.998
6	0.752	0.466	0.0742	0.0214	0.692	0.380	0.958	2.055
7	0.532	0.023	0.0391	0.0008	0.734	0.382	0.957	2.289
8	0.691	0.344	0.0741	0.0174	0.712	0.371	0.957	2.173
9	0.696	0.479	0.0365	0.0147	0.685	0.648	0.991	1.039
10	0.817	0.642	0.0710	0.0285	0.655	0.622	0.979	1.067
11	0.899	0.779	0.1392	0.0547	0.603	0.558	0.945	1.075
12	0.899	0.777	0.1390	0.0543	0.603	0.558	0.944	1.082
13	0.858	0.699	0.1356	0.0521	0.641	0.557	0.956	1.147
14	0.749	0.521	0.0709	0.0258	0.688	0.599	0.981	1.177
15	0.599	0.324	0.0367	0.0118	0.718	0.632	0.983	1.179
16	0.680	0.406	0.0366	0.0118	0.690	0.654	0.988	1.125
17	0.540	0.291	0.0375	0.0131	0.728	0.648	0.970	1.115
18	0.682	0.415	0.0717	0.0238	0.703	0.583	0.973	1.240
19	0.811	0.613	0.1358	0.0501	0.666	0.545	0.962	1.218
20	0.813	0.616	0.1356	0.0500	0.664	0.544	0.961	1.220
21	0.529	0.228	0.0375	0.0098	0.729	0.609	0.970	1.259
22	0.612	0.278	0.0378	0.0092	0.715	0.407	0.974	1.950
23	0.531	0.162	0.0389	0.0066	0.732	0.401	0.961	2.100
24	0.518	0.193	0.0333	0.0074	0.718	0.463	0.945	1.785
25	0.390	0.208	0.0194	0.0080	0.716	0.472	0.925	1.736
26	0.297	0.255	0.0122	0.0098	0.700	0.500	0.891	1.608
27	0.615	0.337	0.0480	0.0153	0.706	0.468	0.947	1.697

Continued on next page

Table D.2: Continued.

Run	$x_{l,h}$	$x_{l,c}$	$\dot{m}_{l,h}$	$\dot{m}_{l,c}$	$\eta_{i,comp}$	$\eta_{i,exp}$	$\eta_{v,comp}$	$\eta_{v,exp}$
-	-	-	kg/s	kg/s	-	-	-	-
28	0.672	0.391	0.0630	0.0197	0.704	0.445	0.950	1.774
29	0.716	0.440	0.0784	0.0244	0.700	0.431	0.948	1.816
30	0.748	0.483	0.0928	0.0291	0.696	0.421	0.948	1.837
31	0.776	0.524	0.1074	0.0342	0.689	0.419	0.945	1.837
32	0.798	0.561	0.1221	0.0394	0.681	0.417	0.941	1.827
33	0.814	0.594	0.1343	0.0447	0.674	0.418	0.937	1.818
34	0.913	0.777	0.1398	0.0466	0.585	0.446	0.911	1.617
35	0.905	0.764	0.1244	0.0424	0.588	0.460	0.913	1.554
36	0.892	0.746	0.1087	0.0386	0.595	0.463	0.922	1.540
37	0.874	0.709	0.0944	0.0332	0.608	0.460	0.933	1.570

Appendix E: Appendices For R410A Flooded Compressor Testing

E.1 Code For Arduino Controller

```
#include <PID_Beta6.h>
#include <AFMotor2.h>
unsigned int buffer[5]={0,0,0,0,0};  // Buffer for Serial reading
float floatbuffer[2]={0,0};          // conversion to float for the PID, which only
    reads float
int readcount=0, speed1=200, speed2=200;     // counter for serial bytes, ms delay
    for motor 1 and 2
unsigned long nextsendmillis=0;  // variables which store the time for the next
    serial action, the next step of motor 1 and the next step of motor 2

// setting up the PID
double Setpoint1, Input1, Output1, P_1 = 1, I_1 = 5, D_1 = 0;
double Setpoint2, Input2, Output2, P_2 = 1, I_2 = 5, D_2 = 0;
PID PID1(&Input1, &Output1, &Setpoint1,P_1,I_1,D_1);
PID PID2(&Input2, &Output2, &Setpoint2,P_2,I_2,D_2);

// setting up the Stepper Valves
AF_Stepper motor1(100, 1); // 3.6 degrees per step = 100 steps/rev (p.7 from Bulletin
     100-20), motor one
AF_Stepper motor2(100, 2); // 3.6 degrees per step = 100 steps/rev (p.7 from Bulletin
     100-20), motor two
int currentstep[2];
boolean Automatic[2]={false, false};
boolean isPositiveActing1=true,isPositiveActing2=true;

void setup()
{
  Serial.begin(9600);    // start serial communication at 9600 bits/second

  Input1 = 0.5;         // initialize with 200 PSI
  Setpoint1 = 0.5;      // initialize setpoint with 200 PSI
  Input2 = 0.5;         // initialize with 200 PSI
  Setpoint2 = 0.5;      // initialize setpoint with 200 PSI

  PID1.SetOutputLimits(0,1596);  // max 1500 steps
  PID1.SetInputLimits(0,1.0);  // input scaled to be within 0 and 1
  PID2.SetOutputLimits(0,1596);  // max 1500 steps
  PID2.SetInputLimits(0,1.0);  // input scaled to be within 0 and 1
  PID1.SetMode(AUTO);
  PID2.SetMode(AUTO);

  motor1.setSpeed(speed1);  // Steps per second
  motor2.setSpeed(speed2);  // Steps per second

  ResetValves();
}
void loop()
{
  if (Serial.available() > 0) {         // if there's data on the serial line, write
      it to the buffer
    buffer[readcount] = Serial.read();
    readcount++;                        // readcount = readcount + 1
  }
  if (readcount>4) {                    // if serial transfer complete (5bytes
      received)
    readcount=0;                        // or buffer full, start deciding what to do
        next
    Serial.flush();                     // clean the internal Serial Buffer
    buffer[1] = (buffer[1]*256) + buffer[2];   // Combine two bytes to a number
        between 0 and 65535
    buffer[3] = (buffer[3]*256) + buffer[4];
    floatbuffer[0] = float(buffer[1]) / 100.0;   // 0 to 655.35
    floatbuffer[1] = float(buffer[3]) / 100.0;   // 0 to 655.35

    if (buffer[0]==0) GetCurrentInput();
    if (buffer[0]==1) GetSetpoints();
    if (buffer[0]==2) ChangeP();
    if (buffer[0]==3) ChangeI();
    if (buffer[0]==4) ChangeD();
    if (buffer[0]==5) SetSpeed();
    if (buffer[0]==6) ResetValves();
    if (buffer[0] > 15) ManualOverride();               //  16 manual override PID 1,
         17 manual override PID 2, 18 manual override PID 1+2, 19 =
        GetCurrentPressures
  }
```

```
  if (Automatic[0]==true) PID1.Compute();       // Let the PID compute the new Output
      Values
  if (Automatic[1]==true) PID2.Compute();
  if (int(Output1) > currentstep[0]) {                  // If the PID spits out new
      Output Values, open or close the valves accordingly
    motor1.step((int(Output1)-currentstep[0]),BACKWARD, DOUBLE);
    currentstep[0]=int(Output1);
  }
  if (int(Output1) < currentstep[0]) {
    motor1.step(currentstep[0]-(int(Output1)),FORWARD, DOUBLE);
    currentstep[0]=int(Output1);
  }
  if (int(Output2) > currentstep[1]) {
    motor2.step((int(Output2)-currentstep[1]),BACKWARD, DOUBLE);
    currentstep[1]=int(Output2);
  }
  if (int(Output2) < currentstep[1]) {
    motor2.step(currentstep[1]-(int(Output2)),FORWARD, DOUBLE);
    currentstep[1]=int(Output2);
  }
  motor1.compute();  // Let the stepper logic do its thing
  motor2.compute();
  if (nextsendmillis < millis()){  // if it's time to send the status, do it!
    StatusSend();
    nextsendmillis += 2000;        // nextstendmills = nextsendmillis + 2000;
                                   // Try to keep this on the order of the time it
                                       takes get input from chip
  }
}
void GetCurrentInput()
{
  Automatic[0]=true;
  Automatic[1]=true;
  // inputs are normalized, come in as 0 to 65535, divided by 100 above, and here by
      655.35
  Input1 = floatbuffer[0]/655.35;  // go from 0 --> 655.35 to 0 --> 1
  Input2 = floatbuffer[1]/655.35; // go from 0 --> 655.35 to 0 --> 1
}
void GetSetpoints()
{
  // setpoints are normalized, come in as 0 to 65535, divided by 100 above, and here
      by 655.35
  Setpoint1 = floatbuffer[0]/655.35;  // go from 0 --> 655.35 to 0 --> 1
  Setpoint2 = floatbuffer[1]/655.35;  // go from 0 --> 655.35 to 0 --> 1
}
void ManualOverride()
{
  if (buffer[0] == 16)        //  16 manual override PID 1
  {
    Automatic[0]=false;
    Automatic[1]=true;
    Output1 = buffer[1];
    Input2 = floatbuffer[1];
  }
  if (buffer[0] == 17)        // 17 manual override PID 2
  {
    Automatic[0]=true;
    Automatic[1]=false;
    Output2 = buffer[3];
    Input1 = floatbuffer[0];
  }
  if (buffer[0] == 18)        // 18 manual override PID 1+2
  {
    Automatic[0]=false;
    Automatic[1]=false;
    Output1 = buffer[1];
    Output2 = buffer[3];
  }
  if (buffer[0] == 19)        // 19 = GetCurrentPressures
  {
    Automatic[0]=true;
    Automatic[1]=true;
    Input1 = floatbuffer[0];
    Input2 = floatbuffer[1];
  }
  if (buffer[0] == 20)        // 20 = Output debug information
  {
      Serial.print("Debug_line:");
```

```
      Serial.print(buffer[0]);
      Serial.println("end");
  }
  if (buffer[0] == 21)
  {
    StatusSend();
  }
  if (buffer[0] == 22)
  {
    if (buffer[2] == 1)
      isPositiveActing1=true;
    else
      isPositiveActing1=false;
    if (buffer[4] == 1)
      isPositiveActing2=true;
    else
      isPositiveActing2=false;
  }
}
void ChangeP()
{
  P_1 = (floatbuffer[0]/200);   // make 0 .. 655.35 into 0 .. 3.2767
  P_2 = (floatbuffer[1]/200);
  if (isPositiveActing1==true)
  {
    P_1=-1*P_1;
  }
  if (isPositiveActing2==true)
  {
    P_2=-1*P_2;
  }
}
void ChangeI()
{
  if (floatbuffer[0]) {  // make sure we don't divide by 0
    I_1 = 100/(floatbuffer[0]);
  } else {
    I_1 =0;
  }
  if (floatbuffer[1]) {
    I_2 = 100/(floatbuffer[1]);
  } else {
    I_2 = 0;
  }
  if (isPositiveActing1==true)
  {
    I_1=-1*I_1;
  }
  if (isPositiveActing2==true)
  {
    I_2=-1*I_2;
  }
}
void ChangeD()
{
  D_1 = floatbuffer[0]/10;
  D_2 = floatbuffer[1]/10;
  if (isPositiveActing1==true)
  {
    D_1=-1*D_1;
  }
  if (isPositiveActing2==true)
  {
    D_2=-1*D_2;
  }
  PID1.SetTunings(P_1, I_1, D_1);
  PID2.SetTunings(P_2, I_2, D_2);
}
void SetSpeed()
{
  speed1=buffer[1];
  motor1.setSpeed(speed1);
  speed2=buffer[3];
  motor2.setSpeed(speed2);
}
void ResetValves()
{
  motor1.step(3500, FORWARD, SINGLE);
  motor2.step(3500, FORWARD, SINGLE);
```

```
  nextsendmillis = millis() + 14100;
  while(nextsendmillis > millis()) // waiting until the reset is complete
  {
    motor1.compute();
    motor2.compute();
  }
  currentstep[0] = 0;
  currentstep[1] = 0;
}
void StatusSend(){
  int SerialBuffer[12]={0,0,0,0,0,0,0,0,0,0,0,0};
  int intSetpoint1,intInput1;
  SerialBuffer[0] = int(Output1);             // extract high byte
  SerialBuffer[0] = SerialBuffer[0] / 256;
  SerialBuffer[1] = int(Output1);             // works because serial only sends the
      low byte of the integer
  SerialBuffer[2] = int(Output2/256);         // extract high byte in another way,
      works too
  SerialBuffer[3] = int(Output2);

  intSetpoint1=Setpoint1*65535; // convert 0-->1 back to 0-->65535
  intInput1=Input1*65535; // convert 0-->1 back to 0-->65535

  SerialBuffer[4] = int(intSetpoint1/256);
  SerialBuffer[5] = int(intSetpoint1);
  SerialBuffer[6] = int(intInput1/256);
  SerialBuffer[7] = int(intInput1);

  SerialBuffer[8] = int(speed1);
  SerialBuffer[9] = int(speed2);
  SerialBuffer[10] = int(isPositiveActing1);
  SerialBuffer[11] = int(isPositiveActing2);

  for (int i=0; i<12; i++) Serial.print(SerialBuffer[i],BYTE);  // sends only the low
       byte of the integer
  Serial.print(10);
}
```

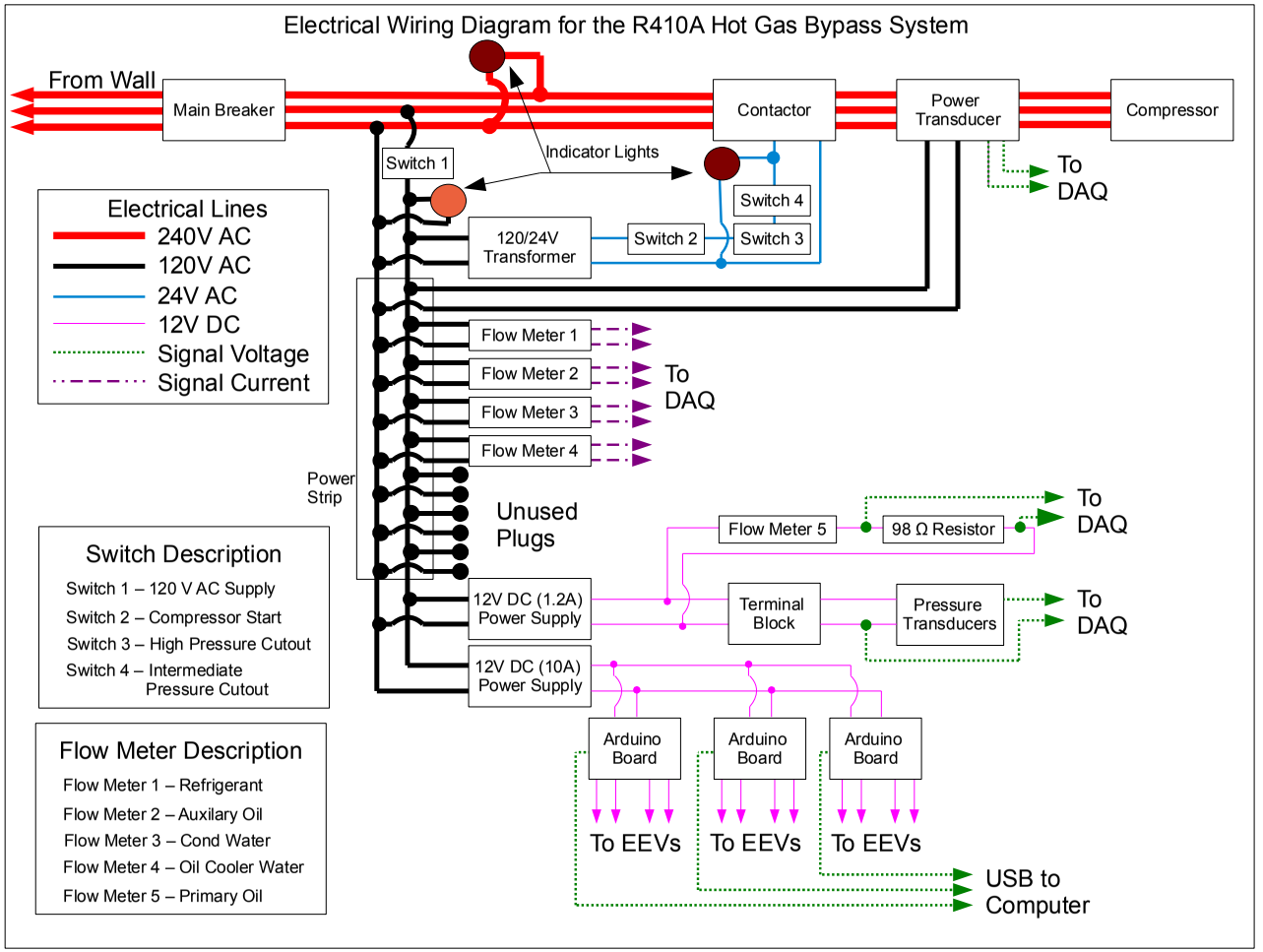
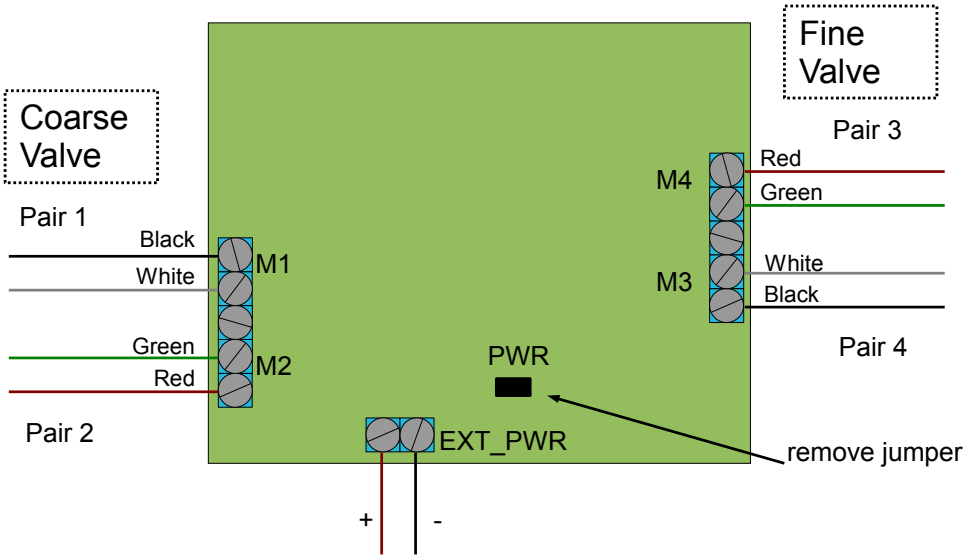


Figure E.1. Wiring schematic of liquid-flooded scroll compressor stand.



http://www.ladyada.net/make/mshield/faq.html

What pins are not used on the motor shield?

All 6 analog input pins are available. They can also be used as digital pins (pins #14 thru 19)

Digital pin 2, and 13 are not used.

The following pins are in use only if the DC/Stepper noted is in use:
Digital pin 11: DC Motor #1 / Stepper #1 (activation/speed control)
Digital pin 3: DC Motor #2 / Stepper #1 (activation/speed control)
Digital pin 5: DC Motor #3 / Stepper #2 (activation/speed control)
Digital pin 6: DC Motor #4 / Stepper #2 (activation/speed control)

The following pins are in use if any DC/steppers are used
Digital pin 4, 7, 8 and 12 are used to drive the DC/Stepper motors via the 74HC595 serial-to-parallel latch

The following pins are used only if that particular servo is in use:
Digitals pin 9: Servo #1 control
Digital pin 10: Servo #2 control

Figure E.2. Wiring schematic of Arduino motor shield.

E.2 Python Code For Data Analysis

```
def p_3MAF_R410A(T,C):
    def func(x, a, b, c):
        return a+b*x+c*x**2
    Cvec=[20,15,10,8,5,2]
    p_2=+0.6465592072+0.0174850997128*T+0.000169331474869*T**2
    p_5=+1.698813952+0.0438996865857*T+0.000372841553778*T**2
    p_8=+2.561011707+0.0726521906849*T+0.000566665868184*T**2
    p_10=+3.125550117+0.0897795222665*T+0.000705521345098*T**2
    p_15=+4.383189772+0.123696033319*T+0.00112111260392*T**2
    p_20=+5.446518446+0.154150414573*T+0.0015490043325*T**2
    pvec=[p_20,p_15,p_10,p_8,p_5,p_2]
    popt, pcov = curve_fit(func, np.array(Cvec), np.array(pvec))
    a=popt[0]
    b=popt[1]
    c=popt[2]
    return (a+b*C+c*C**2)*100.0 #Fits are with pressures in bar
def rho_3MAF_R410A(T,C):
    def func(x, a, b, c):
        return a+b*x+c*x**2
    Cvec=[20,15,10,8,5,2,0]
    ## #Density fit data from 3MAF data sheet
    r_20=-0.000869719897980319*T+1.01505747395872
    r_15=-0.000827907503745965*T+1.00796884424987
    r_10=-0.000792255417964326*T+1.00195564472862
    r_8=-0.000781739765434716*T+0.999804945527956
    r_5=-0.000764208389886381*T+0.9966469957064
    r_2=-0.000751368670331525*T+0.994120917716822
    r_0=-0.0007435116505284 7*T+0.992439584365375
    rhovec=[r_20,r_15,r_10,r_8,r_5,r_2,r_0]
    popt, pcov = curve_fit(func, np.array(Cvec), np.array(rhovec))
    a=popt[0]
    b=popt[1]
    c=popt[2]
    return a+b*C+c*C**2
def Conc_TP(T,p):
    return float(fsolve(lambda C: p_3MAF_R410A(T,C)-p,10))
def f(inputs,**kwargs):
    deriv=dict()
    outputs=dict()
    baseOutputs=kwargs.get('base',None)
    # Get a list of currently defined variables
    # (needed to not write back existing variables)
    definedVariables=dir()
    #unpack variable from dict into variables (not safe)
    exec '\n'.join('%s=%r'%i for i in inputs.items())
    ##################################################################
    ##################### BEGIN USER CODE ############################
    ##################################################################
    rho1=cp.Props('D','T',T1,'P',p1,'R410A')
    h1=cp.Props('H','T',T1,'P',p1,'R410A')
    s1=cp.Props('S','T',T1,'P',p1,'R410A')
    h2=cp.Props('H','T',T2,'P',p2,'R410A')
    h2s=cp.h_sp('R410A',s1,p2,T2)
    if abs(mdot_oil)<0.0001:
        mdot_oil=1e-20
    Vdisp=28e-6
    Vdot=Vdisp*3500/60
    eta_v=mdot_ref/(rho1*Vdot)
    Tsats=cp.Tsat('R410A',p1,1.0,T1)-273.15
    Tsatd=cp.Tsat('R410A',p2,1.0,285)-273.15
```

```
DTsh=(T1-273.15)-Tsats
Tratio=T2/T1
xL=mdot_oil/(mdot_oil+mdot_ref)
Xsep=Conc_TP(T9-273.15,p4)/100.
h11=FP.h_m('R410A','POE',T11,p11,1.0-Xsep)
s11=FP.s_m('R410A','POE',T11,p11,1.0-Xsep)
h2s_inj=FP.h_sp('R410A','POE',s11,p2,1.0-Xsep,T2)
W_dot_i=mdot_ref*(h2s-h1)+mdot_oil*(h2s_inj-h11)
h11=FP.h_m('R410A','POE',T11,p11,1.0-Xsep)
h12=FP.h_m('R410A','POE',T2,p2,1.0-Xsep)
mdotDeltaH=mdot_ref*(h2-h1)+mdot_oil*(h12-h11)
Q_amb=power-mdotDeltaH
eta_io=W_dot_i/power
eta_i=(h2s-h1)/(h2-h1)
mdot_total=mdot_oil+mdot_ref
# ----------------------
#    Condenser Calcs
# ----------------------
T3h=cp.T_hp('R410A',h2,p4,T3)
h3=cp.Props('H','T',T3,'P',p4,'R410A')
h4=cp.Props('H','T',T4,'P',p4,'R410A')
Qcond_ref=mdot_ref*(h3-h4)
cp_w=cp.Props('C','T',(T12+T13)/2.0,'P',200,'REFPROP-Water')
Qcond_w=mdot_w_cond*cp_w*(T12-T13)
# ----------------------
#    Oil cooler calcs
# ----------------------
cp_w=cp.Props('C','T',(T14+T15)/2.0,'P',200,'REFPROP-Water')
c_oil=mdot_w_oilcooler*cp_w*(T15-T14)/(mdot_oil*(T9-T10))
# ----------------------
#   Ambient HT calcs
# ----------------------
Tf_bottom=(TshellBottom+Tamb)/2
Tf_top=(TshellTop+Tamb)/2
rho,c_p,mu,k=AirProps(Tf_top)
nu=mu/rho
alpha=k/(rho*c_p)
Pr=nu/alpha
L=5.5*0.0254
A_top=np.pi*L**2/4.0
Ra_L_top=9.81/Tf_top*(TshellTop-Tamb)**0.0254/(nu*alpha)
Nu_L_top=0.15*Ra_L_top**(1.0/3.0)
h_a_top=Nu_L_top*k/L
e=0.9
sigma=5.67e-8
h_r_top=sigma*e*(TshellTop+Tamb)*(TshellTop**2+Tamb**2)
Q_top=(h_a_top+h_r_top)*A_top*(TshellTop-Tamb)/1000
rho,c_p,mu,k=AirProps(Tf_bottom)
nu=mu/rho
alpha=k/(rho*c_p)
Pr=nu/alpha
L=13.5*0.0254
D=5.5*0.0254
A_bottom=np.pi*D*L
Ra_L_bottom=9.81/Tf_bottom*(TshellBottom-Tamb)*L/(nu*alpha)
Nu_L_bottom=(0.825+0.387*Ra_L_bottom**(1/6.)/(1+(0.492/Pr)
    **(9.0/16.0))**(8.0/27))**2
h_a_bottom=Nu_L_bottom*k/L
e=0.9
sigma=5.67e-8
h_r_bottom=sigma*e*(TshellBottom+Tamb)*(TshellBottom**2+Tamb**2)
Q_bottom=(h_a_bottom+h_r_bottom)*A_bottom*(TshellBottom-Tamb)
    /1000
```

```
    Q_amb_calc=Q_top+Q_bottom
    if not 'base' in kwargs:
        f=pylab.figure()
        ax=f.add_axes((0.15,0.15,0.8,0.8))
        CPPlot.Ph('R410A',axis=ax,bounds='R410A')
        h=[h1,h2,h3,h4]
        p=[p1,p2,p4,p4]
        ax.plot(h,p,'o-')
        pylab.close()
        del h,p,f,ax

    #Remove intermediate variables
    del e,D,L,nu,alpha,rho,c_p,mu,k,Pr,h1,h2,Ra_L_top,Nu_L_top,
        h_a_top,h_r_top
    del s1,h_a_bottom,h_r_bottom,sigma,Ra_L_bottom,Nu_L_bottom
    del Tf_top,Tf_bottom,A_top,A_bottom,h2s,cp_w,T3h,h3,h4,Q_top,
        Q_bottom
    del Qcond_ref,Vdot,Vdisp
    ##########################################################################
    ###################### END USER CODE #####################################
    ##########################################################################
    #pack all calulated variables back into outputs
    for var in dir():
        if notInList(var,definedVariables) and var not in inputs and
            var!='definedVariables':
            outputs[var]=eval(var)
    if 'type' in kwargs and kwargs['type']=='plus' and 'base' in
        kwargs and 'eps' in kwargs:
        for outField in outputs:
            # calculate the numerical derivative of the output term
            deriv[outField]=(outputs[outField]-baseOutputs[outField
                ])/kwargs['eps']
        return deriv
    return outputs

def calcUncertainties(inputs,uncertInput):
    # Make dicts of values and their absolute uncertainties
    absUncertOutput=dict()
    relUncertOutput=dict()
    deriv=dict()
    deriv_temp=dict()
    sumsq=dict()
    epsilon=1e-5
    ## ****************************
    ##            Base code
    ## ****************************
    # Shallow copy the base inputs
    baseInputs=copy.copy(inputs)
    #Call calculation function
    baseOutputs=f(baseInputs)
    ## ****************************
    ##          Uncertainty Code
    ## ****************************
    ## Loop over the inputs
    for inField in baseInputs:
        #shallow copy the inputs
        plusInputs=copy.copy(inputs)
        #increment active field by epsilon
        plusInputs[inField]=plusInputs[inField]+epsilon
        #call calculation function with incremented input to get
            derivatives of the outputs with respect to active field
        deriv[inField]=f(plusInputs,type='plus',eps=epsilon,base=
            baseOutputs)
```

```
#invert the nesting of the dictionaries of derivatives to get
    outputs as a function of inputs
for outField in baseOutputs:
    deriv_temp[outField]=dict()
    for inField in baseInputs:
        deriv_temp[outField][inField]=deriv[inField][outField]
#copy temporary values back to variable deriv
deriv=copy.copy(deriv_temp)
#calculate the absolute and relative uncertanties of each output
    term
for outField in baseOutputs:
    sumsq[outField]=0.0
    for inField in baseInputs:
        sumsq[outField]=sumsq[outField]+(deriv[outField][inField
            ]*uncertInput[inField])**2
    absUncertOutput[outField]=sqrt(sumsq[outField])
    relUncertOutput[outField]=absUncertOutput[outField]/
        baseOutputs[outField]
return baseOutputs,absUncertOutput,relUncertOutput
```

Table E.1: Experimental data from the testing of R410A scroll compressor with oil injection.

#	T_1 °C	T_2 °C	T_3 °C	T_4 °C	T_5 °C	T_6 °C	T_7 °C	T_8 °C	T_9 °C	T_{10} °C	T_{11} °C	T_{12} °C	T_{13} °C	T_{14} °C	T_{15} °C	T_{2B} °C	T_{2C} °C	T_{2D} °C	T_{2E} °C	T_{16} °C	T_{17} °C
1	1.0	81.2	56.3	12.9	13.6	-7.4	-0.1	23.6	24.6	24.7	23.9	12.4	15.1	28.8	28.7	80.4	81.0	79.6	79.7	81.5	39.1
2	1.2	73.4	52.8	12.7	13.5	-6.7	0.3	23.4	48.4	27.9	26.2	12.3	14.8	28.3	28.4	72.7	73.3	72.3	72.4	74.0	36.6
3	1.1	69.9	51.1	12.6	13.4	-7.3	0.2	23.5	53.1	27.1	26.5	12.2	14.5	27.4	27.6	69.4	69.9	69.1	69.1	70.4	36.2
4	1.0	66.0	48.4	12.6	13.4	-6.9	-0.0	23.4	52.1	26.8	26.5	12.1	14.3	27.0	27.4	65.5	66.0	65.3	65.3	66.3	35.6
5	1.0	63.5	46.8	12.6	13.4	-7.0	0.1	23.4	51.6	27.1	26.9	12.0	14.2	27.1	27.6	63.1	63.5	62.9	62.9	63.7	35.6
6	1.0	60.3	44.2	12.6	13.4	-6.3	0.1	23.2	49.4	26.9	26.6	11.9	14.0	26.7	27.4	60.0	60.3	59.7	59.8	60.3	35.2
7	1.1	58.2	42.8	13.2	13.9	-6.9	0.3	23.1	48.1	26.8	26.7	12.4	14.2	26.3	27.1	58.0	58.2	57.7	57.8	58.2	36.2
8	1.1	56.2	41.8	13.6	14.2	-6.7	0.2	23.1	47.0	27.1	27.1	12.6	14.4	26.5	27.4	56.0	56.2	55.8	55.8	56.2	36.1
9	0.9	53.9	40.0	14.5	15.1	-6.9	-0.1	23.1	44.9	27.1	27.1	12.8	14.6	26.3	27.3	53.8	53.9	53.5	53.5	53.9	35.4
10	1.1	52.8	39.2	14.7	15.2	-7.2	0.3	23.2	44.2	27.2	27.3	12.8	14.5	26.3	27.4	52.7	52.8	52.5	52.5	52.8	35.4
11	1.2	116.4	72.6	14.7	15.3	-8.0	-0.6	25.3	19.5	18.4	26.3	12.1	14.9	12.1	12.1	114.9	115.9	113.5	113.6	117.3	55.7
12	1.1	99.6	66.7	13.3	14.0	-7.7	-0.2	25.9	60.2	43.4	32.8	12.9	15.2	34.7	32.3	98.5	99.4	97.9	98.0	100.0	46.5
13	0.9	94.6	63.3	13.4	14.1	-7.6	-0.4	25.7	65.4	54.4	44.2	12.4	15.1	45.8	37.5	93.7	94.5	93.1	93.3	95.5	43.7
14	1.1	93.2	63.2	13.4	14.2	-7.2	-0.1	26.1	68.1	59.9	50.6	12.5	15.2	50.1	40.1	92.3	93.2	91.8	92.0	94.3	44.5
15	1.1	91.8	63.2	13.5	14.2	-7.6	-0.2	26.3	69.5	63.2	54.8	12.5	15.2	52.6	41.5	91.0	91.8	90.6	90.7	92.8	45.6

Continued on next page

Table E.1: Continued.

#	T_1 °C	T_2 °C	T_3 °C	T_4 °C	T_5 °C	T_6 °C	T_7 °C	T_8 °C	T_9 °C	T_{10} °C	T_{11} °C	T_{12} °C	T_{13} °C	T_{14} °C	T_{15} °C	T_{2B} °C	T_{2C} °C	T_{2D} °C	T_{2E} °C	T_{16} °C	T_{17} °C
16	1.2	90.3	63.4	13.5	14.3	-7.3	-0.0	26.5	70.8	65.9	58.9	12.5	15.3	55.6	43.0	89.7	90.4	89.2	89.3	91.2	46.3
17	1.3	89.4	63.5	13.5	14.3	-7.4	0.1	26.6	71.5	67.6	61.4	12.5	15.3	57.2	43.9	89.0	89.5	88.4	88.5	90.0	46.9
18	1.0	88.8	63.8	13.5	14.3	-7.3	-0.2	26.7	72.3	69.1	63.5	12.6	15.4	58.6	45.2	88.4	88.8	87.8	87.9	89.1	48.6
19	1.4	88.0	64.1	13.7	14.5	-7.5	0.1	26.8	73.1	70.4	65.6	12.8	15.6	59.9	46.4	87.8	88.1	87.1	87.2	88.2	49.3
20	1.1	87.5	64.5	13.6	14.4	-7.3	-0.1	26.7	73.6	71.4	67.2	12.7	15.6	61.1	47.5	87.3	87.5	86.6	86.7	87.6	49.0
21	1.1	74.3	52.4	15.0	15.8	-7.7	-0.1	26.3	58.7	35.0	34.8	12.7	15.1	34.2	35.4	74.1	74.4	73.7	73.7	74.5	42.7
22	1.2	78.2	54.6	14.6	15.3	-7.6	0.1	26.3	61.0	35.3	34.9	12.7	15.2	34.8	35.7	77.8	78.2	77.4	77.5	78.6	42.8
23	1.2	81.8	57.1	14.3	15.0	-7.5	0.0	26.7	63.2	35.5	35.0	12.7	15.3	35.4	36.1	81.3	81.8	80.9	81.0	82.4	43.0
24	1.0	85.4	59.3	14.0	14.7	-6.9	-0.3	26.8	64.9	35.5	34.8	12.7	15.4	35.6	36.2	84.8	85.4	84.4	84.5	86.2	43.4
25	1.2	89.9	62.0	13.7	14.5	-7.3	-0.0	27.0	66.5	36.4	35.0	12.6	15.4	36.7	37.1	89.1	89.8	88.7	88.8	90.6	44.4
26	1.1	94.2	64.3	13.5	14.2	-7.6	-0.2	27.1	66.4	37.3	34.9	12.5	15.3	37.7	37.9	93.2	94.0	92.8	92.9	94.8	44.9
27	4.9	80.2	55.3	14.0	14.8	-7.0	3.9	26.7	62.6	35.6	35.0	12.3	14.6	35.1	36.1	79.8	80.2	79.3	79.4	80.7	45.0
28	15.1	85.4	56.7	13.7	14.8	-3.5	14.9	27.0	66.5	35.7	35.0	12.3	14.2	35.1	36.1	85.0	85.5	84.3	84.5	86.0	53.0
29	1.2	111.2	70.5	13.0	13.8	-7.7	-0.6	27.1	28.5	25.9	26.4	12.2	14.9	26.5	28.5	109.9	110.8	108.6	108.7	111.8	50.2
30	1.1	97.4	62.4	12.5	13.3	-7.3	-0.3	23.1	55.5	12.4	18.1	12.0	15.2	12.1	12.7	96.3	97.2	95.6	95.7	97.9	43.2

Continued on next page

Table E.1: Continued.

#	T_1 °C	T_2 °C	T_3 °C	T_4 °C	T_5 °C	T_6 °C	T_7 °C	T_8 °C	T_9 °C	T_{10} °C	T_{11} °C	T_{12} °C	T_{13} °C	T_{14} °C	T_{15} °C	T_{2B} °C	T_{2C} °C	T_{2D} °C	T_{2E} °C	T_{16} °C	T_{17} °C
31	1.4	90.4	60.1	12.5	13.3	-7.1	0.0	23.6	62.4	12.4	15.8	12.0	15.1	12.1	13.4	89.4	90.3	89.1	89.1	91.0	42.2
32	1.1	85.1	57.2	12.6	13.4	-7.3	-0.4	24.7	61.9	13.0	15.5	12.0	14.7	12.1	13.9	84.3	85.0	83.9	84.0	85.9	41.5
33	1.1	79.6	52.0	12.9	13.6	-7.6	-0.3	24.4	58.5	12.8	15.3	11.9	14.3	11.9	14.3	78.9	79.6	78.5	78.6	80.4	39.8
34	1.1	75.2	48.9	13.3	14.0	-7.5	-0.4	23.9	55.5	13.5	15.4	12.0	14.2	12.0	14.7	74.7	75.2	74.3	74.4	75.8	39.5
35	1.1	69.4	44.8	14.7	15.1	-7.2	-0.2	22.4	51.2	14.8	16.0	12.6	14.4	12.6	15.8	69.1	69.4	68.6	68.7	69.7	39.2

Continued on next page

Table E.1: Continued.

#	T_{amb}	p_1	p_2	p_4	p_5	p_7	p_8	p_{11}	$\dot{m}_{ref}$	$\dot{m}_l$	$\dot{m}_{w,cond}$	$\dot{m}_{w,oc}$	ρ_{oil}	$\dot{W}_{\text{el}}$	$T_{dew,s}$	$T_{dew,d}$	ΔT_{sh}	x_l	η_{oi}
	°C	kPa	kPa	kPa	kPa	kPa	kPa	kPa	g/s	g/s	g/s	g/s	kg/m^3	kW	°C	°C	K	-	-
1	24.0	573	1885	1397	1410	626	1320	669	30.08	-0.09	127.15	150.60	1004.7	1.781	-10.0	30.0	11.0	0.000	0.597
2	23.8	574	1882	1395	1406	631	1312	677	31.37	1.63	133.59	156.94	1004.4	1.727	-9.9	30.0	11.1	0.049	0.645
3	24.3	573	1884	1378	1386	629	1310	694	31.32	3.37	131.85	153.75	1004.9	1.727	-10.0	30.0	11.2	0.097	0.651
4	24.1	571	1875	1352	1359	628	1350	720	31.34	5.57	130.93	152.53	1006.0	1.721	-10.1	29.8	11.0	0.151	0.658
5	24.3	573	1880	1338	1343	630	1340	743	31.45	7.70	130.68	152.16	1006.0	1.725	-10.0	29.9	11.0	0.197	0.664
6	24.0	573	1885	1318	1323	630	1334	737	31.57	10.76	129.34	149.92	1007.9	1.731	-10.0	30.0	11.0	0.254	0.675
7	24.2	573	1887	1316	1321	630	1329	742	31.55	13.85	136.61	157.02	1008.4	1.759	-10.0	30.1	11.1	0.305	0.674
8	24.0	573	1884	1317	1321	630	1327	765	31.57	17.00	136.25	156.73	1008.9	1.758	-10.0	30.0	11.1	0.350	0.683
9	24.4	571	1877	1315	1317	628	1323	804	31.55	20.98	135.42	155.18	1010.0	1.753	-10.1	29.9	11.0	0.399	0.693
10	24.3	573	1881	1310	1313	630	1322	823	31.66	23.92	135.19	154.96	1010.3	1.760	-10.0	29.9	11.1	0.430	0.700
11	24.6	574	2620	1336	1344	619	1342	836	26.34	-0.09	146.39	116.56	1005.1	2.402	-9.9	43.3	11.1	0.000	0.500
12	26.2	573	2619	1415	1421	625	1423	684	29.62	1.60	173.63	0.04	995.8	2.305	-10.0	43.3	11.1	0.051	0.592
13	25.4	573	2615	1365	1372	628	1317	726	29.94	3.56	150.71	0.04	995.8	2.293	-10.0	43.2	10.9	0.106	0.605
14	25.4	573	2619	1367	1372	626	1325	774	29.76	5.37	147.86	0.04	981.0	2.308	-10.0	43.3	11.0	0.153	0.603
15	25.6	573	2620	1367	1372	626	1330	810	29.72	7.25	148.20	0.04	978.8	2.311	-10.0	43.3	11.1	0.196	0.606

Continued on next page

Table E.1: Continued.

#	T_{amb}	p_1	p_2	p_4	p_5	p_7	p_8	p_{11}	$\dot{m}_{ref}$	$\dot{m}_l$	$\dot{m}_{w,cond}$	$\dot{m}_{w,oc}$	ρ_{oil}	$\dot{W}_{\text{el}}$	$T_{dew,s}$	$T_{dew,d}$	ΔT_{sh}	x_l	η_{oi}
	°C	kPa	kPa	kPa	kPa	kPa	kPa	kPa	g/s	g/s	g/s	g/s	kg/m^3	kW	°C	°C	K	-	-
16	25.9	572	2615	1371	1374	626	1334	839	29.77	9.96	144.90	0.04	974.3	2.310	-10.0	43.2	11.2	0.251	0.613
17	26.1	575	2625	1373	1375	628	1337	859	29.88	12.82	143.93	0.04	972.7	2.349	-9.9	43.4	11.2	0.300	0.611
18	26.6	571	2615	1375	1377	625	1340	871	29.70	15.89	143.83	0.04	971.1	2.357	-10.1	43.2	11.1	0.349	0.613
19	26.7	574	2614	1383	1384	628	1342	890	29.86	19.62	143.07	0.04	969.5	2.360	-9.9	43.2	11.3	0.396	0.622
20	26.7	572	2616	1383	1384	624	1345	903	29.71	24.09	142.73	0.04	968.3	2.369	-10.0	43.3	11.2	0.448	0.628
21	27.8	571	2617	1337	1335	625	1350	814	30.39	17.35	135.32	159.05	997.2	2.312	-10.1	43.3	11.2	0.363	0.646
22	28.0	573	2620	1344	1340	627	1352	778	30.44	13.14	139.21	158.88	996.3	2.317	-10.0	43.3	11.2	0.301	0.634
23	28.1	573	2619	1355	1349	627	1356	806	30.41	10.01	139.87	157.72	995.5	2.314	-10.0	43.3	11.1	0.248	0.625
24	28.1	571	2618	1364	1357	626	1360	781	30.20	7.55	139.65	157.14	994.3	2.314	-10.1	43.3	11.1	0.200	0.616
25	28.4	573	2619	1371	1362	627	1366	763	30.11	5.21	139.86	161.25	993.9	2.309	-10.0	43.3	11.2	0.148	0.609
26	28.8	572	2619	1378	1367	626	1369	734	29.96	3.33	140.81	157.69	993.2	2.308	-10.0	43.3	11.1	0.100	0.602
27	27.3	573	2617	1325	1312	627	1372	834	29.96	12.75	136.51	163.08	995.3	2.314	-10.0	43.3	14.9	0.299	0.634
28	26.7	572	2618	1301	1287	627	1373	826	28.66	12.40	135.95	162.19	993.8	2.329	-10.0	43.3	25.1	0.302	0.635
29	26.6	571	2615	1365	1350	618	1376	695	27.99	-0.09	158.01	0.04	993.8	2.353	-10.1	43.2	11.3	0.000	0.545
30	22.4	572	2620	1392	1403	626	1239	726	29.78	1.58	126.85	60.60	1008.6	2.293	-10.0	43.3	11.1	0.050	0.598

Continued on next page

Table E.1: Continued.

#	T_{amb}	p_1	p_2	p_4	p_5	p_7	p_8	p_{11}	$\dot{m}_{ref}$	$\dot{m}_l$	$\dot{m}_{w,cond}$	$\dot{m}_{w,oc}$	ρ_{oil}	$\dot{W}_{\text{el}}$	$T_{dew,s}$	$T_{dew,d}$	ΔT_{sh}	x_l	η_{oi}
	°C	kPa	kPa	kPa	kPa	kPa	kPa	kPa	g/s	g/s	g/s	g/s	kg/m^3	kW	°C	°C	K	-	-
31	24.3	575	2619	1379	1387	630	1245	751	30.28	3.47	126.09	60.28	1009.9	2.298	-9.9	43.3	11.2	0.103	0.608
32	25.3	572	2616	1359	1363	627	1258	773	30.22	5.38	134.61	64.67	1011.0	2.300	-10.0	43.3	11.1	0.151	0.611
33	23.0	572	2617	1330	1334	628	1261	795	30.45	7.64	137.69	66.33	1012.2	2.304	-10.0	43.3	11.1	0.201	0.618
34	22.5	572	2617	1316	1319	628	1261	813	30.61	10.20	144.53	70.12	1013.6	2.318	-10.0	43.3	11.1	0.250	0.622
35	20.9	573	2620	1315	1319	629	1256	846	30.83	14.68	149.77	72.69	1015.6	2.314	-10.0	43.3	11.1	0.323	0.638

Appendix F: Scroll Compressor Model Code

Code Listing

- StructsMacros.h
- CompressorModel.c
- LoadInputs.h
- LoadInputs.c
- SaveOutputs.h
- SaveOutputs.c
- geoFuncs.h
- geoFuncs.c
- MassFlow.h
- MassFlow.c
- HeatTransfer.h
- HeatTransfer.c
- ScrollsModel.h
- ScrollsModel.c
- FloodProp.h
- FloodProp.c
- R744.h
- R744.c
- R410A.h
- R410A.c
- MyFuncs.h
- MyFuncs.c

StructsMacros.h

```
#ifndef _STRUCTS_MACROS_H
#define _STRUCTS_MACROS_H
   #define Ntheta_MAX 50000
   #define T_p_xL 0
   #define T_m_xL 1
   #define true 1
   #define false 0
   #define NTHETA_ADISC 100
   // ***************************************************************
   //                      Structure Prototypes
   // ***************************************************************
   struct phiVals{
      double phi_fi0;
      double phi_fis;
      double phi_fie;
      double phi_fo0;
      double phi_fos;
      double phi_foe;
      double phi_oi0;
      double phi_ois;
      double phi_oie;
      double phi_oo0;
      double phi_oos;
      double phi_ooe;
   };
   struct discVals{
      double x0;
      double y0;
      double R;
      double Cd;
```

```
    double xa_arc1;
    double ya_arc1;
    double ra_arc1;
    double t1_arc1;
    double t2_arc1;
    double m_line;
    double b_line;
    double t1_line;
    double t2_line;
    double xa_arc2;
    double ya_arc2;
    double ra_arc2;
    double t1_arc2;
    double t2_arc2;
    char Type[100];
    double thetaAdisc[NTHETA_ADISC];
    double Adisc[NTHETA_ADISC];
};
struct suctVals{
    double A_sa_suction;
};
struct wallVals{
    double x0;
    double y0;
    double r;
};
struct flowModelVals{
    int flank;
    int radial;
    int s_sa;
    int suction;
    int d_dd;
    int discharge;
};
struct geoVals{
    struct phiVals phi;
    struct discVals disc;
    struct suctVals suct;
    struct wallVals wall;
    struct flowModelVals flowModels;
    double rb;
    double ro;
    double t;
    double hs;
    double delta_flank;
    double delta_radial;
};

struct flowVecVals{
    int *CV1;
    int *CV2;
    int *CVup;
    double *A;
    int *flowModel;
    double *mdot;
    double *mdot_L;
    double *mdot_g;
    double *mdot_c;
    double *p_up;
    double *p_down;
    double *h_up;
    double *h_down;
    double *T_up;
    double *T_down;
    double *xL;
    double *Ed;
    double *Re;
    double *Ma;
    int N;
};
struct scrollInputVals
{
    double T_in;
    double T_out;
    double T_amb;
    double p_in;
    double p_out;
    double xL_in;
    char Ref[200];
    char Liq[200];
    double omega;
};
struct ExperVals
```

```
{
    double mdot;
    double T_d;
    double P_shaft;
    double eta_c;
    double eta_v;
};
struct FlowVals
{
    double w_ent;
    double sigma; //Area ratio used for nozzle model
    double Z_D_bends;
    double L_inlet;
    double D_inlet; // Inlet and outlet ports are assumed to be same diameter and
        length
    double L_flank;
    double delta_flank;
    double delta_radial;
    double phi_flank;
    int flowModel_flank;
    int flowModel_radial;
    int flowModel_s_sa;
    int flowModel_d_dd;
    int flowModel_suction;
    int flowModel_discharge;
    double Cd_inlet;
};
struct MLVals
{
    double m;
    double b;
    double c;
    double eta_m;
    char Type[200];
    double UA_amb;
    double etac_guess;
};
struct flagVals
{
    int useDDD;
    int LeftDischarge; // At theta just to the left of the discharge angle
    int lastRotation;
};
struct PowerEffVals
{
    double P_gas;
    double P_shaft;
    double P_ML;
    double eta_m;
    double eta_c;
    double eta_v;
};
struct HTVals
{
    double T_scroll;
    double T_amb;
    double *hc;
    double *A_wall_i;
    double *A_wall_o;
    double *Tm_plate;
    double *Tm_wall_i;
    double *Tm_wall_o;
    double Q_scroll_gas;
    double Q_scroll_inlet;
    double Q_scroll_outlet;
    double Q_scroll_plenum;
    double Q_scroll_amb;
};
struct massFlowVals
{
    struct FlowVals Inputs;
    double mdot_tot;
    double mdot_suct;
    double mdot_disc;
};
struct DebugVals
{
    double wrap_error_rel;
    double mdot_error_abs;
    double LumpHT_error_abs;
    double Td_error_abs;
    int Ntheta;
    double ElapsedTime;
};
```

```
    struct LossesVals
    {
       double suction;
       double discharge;
       double leakage_flank;
       double leakage_radial;
       double mechanical;
       double Wdot_adiabatic;
    };
    struct solverVals
    {
       double Td_halfBandWidth;
    };
    struct ForcesVals
    {
       double *Fx;
       double *Fy;
       double *Fz;
       double *Mx;
       double *My;
       double *Mz;
       double *MO;
       double *xcp;
       double *ycp;
       double *sumFx;
       double *sumFy;
       double *sumFz;
       double *sumMx;
       double *sumMy;
       double *sumMz;
       double *sumMO;
       double *tau;
       double *Frad;
       double Fx_mean;
       double Fy_mean;
       double Fz_mean;
       double Mx_mean;
       double My_mean;
       double Mz_mean;
       double MO_mean;
       double tau_mean;
       double Frad_mean;
    };
    struct scrollVals
    {
       struct solverVals solver;
       struct geoVals geo;
       struct flagVals flags;
       struct HTVals HT;
       struct MLVals ML;
       struct massFlowVals massFlow;
       struct flowVecVals *flowVec;
       struct scrollInputVals Inputs;
       struct ExperVals Exper;
       struct PowerEffVals PowerEff;
       struct DebugVals Debug;
       struct LossesVals Losses;
       struct ForcesVals Forces;
       double *V;
       double *dV;
       double *T;
       double *p;
       double *xL;
       double *m;
       double *Q;
       double *rho;
       double *error;
       double *theta;
       char Ref[200];                /* The working fluid   */
       char Liq[200];                /* The flooding liquid    */
       double omega;
       int N;
       int Ntheta;
    };
#endif
#ifdef HUGE_VAL
   #define _HUGE HUGE_VAL
#else
   // GCC Version of huge value macro
   #ifdef HUGE
   #define _HUGE HUGE
   #endif
```

```
#endif
```

CompressorModel.c

```
#ifndef __GNUC__
#define _CRTDBG_MAP_ALLOC
#define _CRT_SECURE_NO_WARNINGS
#include <stdlib.h>
#include <crtdbg.h>
#endif
#include <stdio.h>
#include <math.h>
#include <time.h>
#include <string.h>
#include "sys/stat.h"
#include "StructsMacros.h"
#include "Geo/geoFuncs.h"
#include "Props/FloodProp.h"
#include "Props/R134a.h"
#include "Props/R410A.h"
#include "Props/R290.h"
#include "Props/R744.h"
#include "MyFuncs.h"
#include "LoadInputs.h"
#include "MassFlow.h"
#include "ScrollsModel.h"
#include "Geo/gpc.h"
#include "SaveOutputs.h"
#include "CompressorModel.h"
#define printf printf_plus
void parseArgs(int Narg, char *argv[])
{
   /*
   Parse the command-line arguments to ScrollCompressorModel executable.
   Possible arguments and default parameter in [] if applicable:
   -I ["Inputs/Default"]  :: Path to folder with the input files to be used
   -M ["Terminal-File"] :: Display Mode: Whether to write output both to a log fle
       and the terminal or not
   -D :: Description of the batch (will be appended to folder for run)
   -O ["../../ScrollModel_Results"] :: Path to the base folder for output from model
       (relative paths are acceptable)
   Sample call of executable:
   ScrollCompressorModel -I "Inputs/R410A" -D "Points for R410A" -O ".."
   */
   int i;
   int LoadedInputs=false,LoadedDescrip=false,LoadedDisplayMode=false,
       LoadedOutputFolder=false;
   printf_plus("You just fired up %s...\n",argv[0]);
   // Starts at 1 since 0 is the path and name of executable
   for (i=1;i<Narg;i++)
   {
      /* Load up the path which points to the folder where the inputs are */
      if (!strcmp(argv[i],"-I"))
      {
         if (strlen(argv[i+1])<400)
         {
            // Path to the folder with the files which define the job to be run
            strcpy(InputPath,argv[i+1]);
            printf_plus("Input Path is : %s\n",InputPath);
         }
         else
         {
            printf_plus("Input path too long (>400 chars)... Exiting");
         }
         LoadedInputs=true;
      }
      if (!strcmp(argv[i],"-M"))
      {
         if (strlen(argv[i+1])<100)
         {
            // Display model
            strcpy(DisplayMode,argv[i+1]);
            printf_plus("Display Mode is : %s\n",DisplayMode);
         }
         else
```

```
            {
               printf_plus("Input display mode too long (>100 chars)... Exiting");
            }
            LoadedDisplayMode=true;
         }
         if (!strcmp(argv[i],"-O"))
         {
            if (strlen(argv[i+1])<100)
            {
               // Path to the folder with the files which define the job to be run
               strcpy(baseOutputFolder,argv[i+1]);
               printf_plus("Display Mode is : %s\n",baseOutputFolder);
            }
            else
            {
               printf_plus("Output Folder too long (>100 chars)... Exiting");
            }
            LoadedOutputFolder=true;
         }
         if (!strcmp(argv[i],"-D"))
         {
            if (!strcmp(argv[i+1],"ask"))
            {
               fputs("Batch Description [100 char max]: ", stdout);
               fflush(stdout);
               if ( fgets(batchDescription, sizeof batchDescription, stdin) != NULL )
               {
                 char *newline = strchr(batchDescription, '\n'); /* search for newline
                     character */
                 if ( newline != NULL )
                 {
                    *newline = '\0'; /* overwrite trailing newline */
                 }
                 printf_plus("Batch Description = \"%s\"\n", batchDescription);
               }
            }
            else
            {
               strcpy(batchDescription,argv[i+1]);
            }
            LoadedDescrip=true;
         }
      }
      // Load up default parameters
      if (LoadedInputs==false)
      {
         strcpy(InputPath,"Inputs/Default");
         printf_plus("No input path provided; default [Inputs/Default] being used...\n\n
             ");
      }
      if (LoadedDisplayMode==false)
      {
         printf_plus("No display mode provided; default [Terminal-File] being used...\n\
             n");
         strcpy(DisplayMode,"Terminal-File");
      }
      if (LoadedOutputFolder==false)
      {
         #if defined(__WIN32__) || defined(__WIN64__) || defined(_MSC_VER)
            strcpy(baseOutputFolder,"..\\..\\ScrollModel_Results");
         #else
            strcpy(baseOutputFolder,"../../ScrollModel_Results");
         #endif
      }
      if (LoadedDescrip==false)
      {
         fputs("Batch Description [100 char max]: ", stdout);
         fflush(stdout);
         if ( fgets(batchDescription, sizeof batchDescription, stdin) != NULL )
         {
           char *newline = strchr(batchDescription, '\n'); /* search for newline
               character */
           if ( newline != NULL )
              *newline = '\0'; /* overwrite trailing newline */
         }
      }
   }
   int main (int Narg, char *argv[])
   {
      // Main function for the model
```

```
int i,j,k,l,m;
struct geoVals geo;
struct scrollVals scroll;
struct ExperVals Exper;
struct scrollInputVals scrollInput;
struct InputVals Inputs;
struct MLVals ML;
struct FlowVals Flows;
// Parse the command line arguments with the function above
parseArgs(Narg,argv);
// Reset run counter
runNumber=1;
// Load the structures with the input parameters
LoadInputs(&Inputs);
for (i=0;i<Inputs.Ngeo;i++)
{
   for (k=0;k<Inputs.Ndisc;k++)
   {
      for (l=0;l<Inputs.NML;l++)
      {
         for (m=0;m<Inputs.Nflows;m++)
         {
            for (j=0;j<Inputs.Nstate;j++)
            {
               //Load up geo
               geo.rb=            Inputs.geoInput[i].rb;
               geo.hs=            Inputs.geoInput[i].hs;
               geo.phi.phi_fi0=   Inputs.geoInput[i].phi_fi0;
               geo.phi.phi_fis=   Inputs.geoInput[i].phi_fis;
               geo.phi.phi_fie=   Inputs.geoInput[i].phi_fie;
               geo.phi.phi_fo0=   Inputs.geoInput[i].phi_fo0;
               geo.phi.phi_fos=   Inputs.geoInput[i].phi_fos;
               geo.phi.phi_foe=   Inputs.geoInput[i].phi_foe;
               geo.phi.phi_oi0=   Inputs.geoInput[i].phi_oi0;
               geo.phi.phi_ois=   Inputs.geoInput[i].phi_ois;
               geo.phi.phi_oie=   Inputs.geoInput[i].phi_oie;
               geo.phi.phi_oo0=   Inputs.geoInput[i].phi_oo0;
               geo.phi.phi_oos=   Inputs.geoInput[i].phi_oos;
               geo.phi.phi_ooe=   Inputs.geoInput[i].phi_ooe;
               geo.delta_flank=   Inputs.FlowInput[m].delta_flank;
               geo.delta_radial=  Inputs.FlowInput[m].delta_radial;
               geo.t=             geo.rb*(geo.phi.phi_fi0-geo.phi.phi_fo0);
               geo.ro=            geo.rb*PI-geo.t;
               geo.disc.xa_arc1 =    Inputs.discInput[k].xa_arc1;
               geo.disc.ya_arc1 =    Inputs.discInput[k].ya_arc1;
               geo.disc.ra_arc1 =    Inputs.discInput[k].ra_arc1;
               geo.disc.t1_arc1 =    Inputs.discInput[k].t1_arc1;
               geo.disc.t2_arc1 =    Inputs.discInput[k].t2_arc1;
               geo.disc.m_line  =    Inputs.discInput[k].m_line;
               geo.disc.b_line  =    Inputs.discInput[k].b_line;
               geo.disc.t1_line =    Inputs.discInput[k].t1_line;
               geo.disc.t2_line =    Inputs.discInput[k].t2_line;
               geo.disc.xa_arc2 =    Inputs.discInput[k].xa_arc2;
               geo.disc.ya_arc2 =    Inputs.discInput[k].ya_arc2;
               geo.disc.ra_arc2 =    Inputs.discInput[k].ra_arc2;
               geo.disc.t1_arc2 =    Inputs.discInput[k].t1_arc2;
               geo.disc.t2_arc2 =    Inputs.discInput[k].t2_arc2;
               strcpy(geo.disc.Type,Inputs.discInput[k].Type);
               geo.disc.x0=       Inputs.discInput[k].disc_x0;
               geo.disc.y0=       Inputs.discInput[k].disc_y0;
               geo.disc.R=     Inputs.discInput[k].disc_R;
               geo.disc.Cd=       Inputs.discInput[k].disc_Cd;
               geo.suct.A_sa_suction=Inputs.geoInput[i].A_sa_suction*Inputs.
                   FlowInput[m].Cd_inlet;
               printf("Assa %g\n",geo.suct.A_sa_suction);
               geo.wall.r=        Inputs.geoInput[i].wall_r;
               setDiscGeo(&geo,geo.disc.Type);
               // Copy over state inputs
               scrollInput.T_in= Inputs.stateInput[j].T_in;
               scrollInput.T_amb=    Inputs.stateInput[j].T_amb;
               scrollInput.p_in= Inputs.stateInput[j].p_in;
               scrollInput.p_out=    Inputs.stateInput[j].p_out;
               scrollInput.xL_in=    Inputs.stateInput[j].xL_in;
```

```
                    scrollInput.omega=   Inputs.stateInput[j].omega;
                    strcpy(scrollInput.Ref,Inputs.stateInput[j].Ref);
                    strcpy(scrollInput.Liq,Inputs.stateInput[j].Liq);
                    // Copy over experimental values (if applicable)
                    Exper.P_shaft  =Inputs.stateInput[j].Power_exper;
                    Exper.T_d      =Inputs.stateInput[j].Td_exper;
                    Exper.mdot     =Inputs.stateInput[j].mdot_exper;
                    Exper.eta_c    =Inputs.stateInput[j].etac_exper;
                    Exper.eta_v    =Inputs.stateInput[j].etav_exper;
                    // Copy over mechanical inputs
                    ML = Inputs.MLInput[l];
                    // Copy over flow parameters
                    Flows = Inputs.FlowInput[m];
                    geo.flowModels.suction=Inputs.FlowInput[m].flowModel_suction;
                    geo.flowModels.discharge=Inputs.FlowInput[m].flowModel_discharge;
                    geo.flowModels.d_dd=Inputs.FlowInput[m].flowModel_d_dd;
                    geo.flowModels.s_sa=Inputs.FlowInput[m].flowModel_s_sa;
                    geo.flowModels.flank=Inputs.FlowInput[m].flowModel_flank;
                    geo.flowModels.radial=Inputs.FlowInput[m].flowModel_radial;
                    // Run the scroll model using the parameters loaded for this run
                    scroll=Initialize_ScrollModel(&geo,&scrollInput,&Exper,&ML, &Flows)
                        ;
                    // Clear the data for the scroll wraps
                    freeScroll(&scroll);
                    cleanUpGeo();
                    // Increment run counter
                    runNumber++;
                }
            }
        }
    }
   }
   // Free memory allocated for input structure storage
   freeInputs(&Inputs);
   // If running in Visual Studio, this command will print out memory leaks
   // Hopefully it will do nothing since there are no leaks
   #ifndef __GNUC__
   _CrtDumpMemoryLeaks();
   #endif
  return 0;
}
```

LoadInputs.h

```
// loadInputs.h
#ifndef LOAD_INPUTS
#define LOAD_INPUTS

char InputPath[400];
char batchDescription[100];

struct rowVals{
   char strings[300][200];
};
struct stateInputVals{
   char Ref[100];
   char Liq[100];
   double T_in;
   double T_amb;
   double p_in;
   double p_out;
   double xL_in;
   double mdot_exper;
   double Td_exper;
   double Power_exper;
   double etac_exper;
   double etav_exper;
   double omega;
};
struct geoInputVals{
   double rb;
   double hs;
   double phi_fi0;
   double phi_fis;
   double phi_fie;
```

```
    double phi_fo0;
    double phi_fos;
    double phi_foe;
    double phi_oi0;
    double phi_ois;
    double phi_oie;
    double phi_oo0;
    double phi_oos;
    double phi_ooe;
    double delta_flank;
    double delta_radial;
    double xa_arc1;
    double ya_arc1;
    double ra_arc1;
    double t1_arc1;
    double t2_arc1;
    double m_line;
    double b_line;
    double t1_line;
    double t2_line;
    double xa_arc2;
    double ya_arc2;
    double ra_arc2;
    double t1_arc2;
    double t2_arc2;
    double disc_x0;
    double disc_y0;
    double disc_R;
    double A_sa_suction;
    double wall_r;
};
struct discInputVals{
    char Type[100];
    double xa_arc1;
    double ya_arc1;
    double ra_arc1;
    double t1_arc1;
    double t2_arc1;
    double m_line;
    double b_line;
    double t1_line;
    double t2_line;
    double xa_arc2;
    double ya_arc2;
    double ra_arc2;
    double t1_arc2;
    double t2_arc2;
    double disc_x0;
    double disc_y0;
    double disc_R;
    double disc_Cd;
};
struct InputVals{
    struct geoInputVals *geoInput;
    int Ngeo;
    struct stateInputVals *stateInput;
    int Nstate;
    struct discInputVals *discInput;
    int Ndisc;
    struct MLVals * MLInput;
    int NML;
    struct FlowVals * FlowInput;
    int Nflows;
};
/* ------------------------------------------------------------ */
/*                Function Prototypes                           */
/* ------------------------------------------------------------ */
int LoadInputs(struct InputVals *Inputs);
void freeInputs(struct InputVals *Inputs);
struct rowVals * LoadFile(char *fileName, int *nC, int *nR);
#endif
```

LoadInputs.c

```
/*
 *  loadInputs.c
 *
 *  The functions in this file are used to load parameters from csv-files as well
```

```
 *  as validate that the values that have been read in are acceptable
 */
#ifndef __GNUC__
#define _CRT_SECURE_NO_WARNINGS
#define _CRTDBG_MAP_ALLOC
#include <stdlib.h>
#include <crtdbg.h>
#endif
#include <stdio.h>
#include <stdlib.h>
#include "StructsMacros.h"
#include "string.h"
#include "LoadInputs.h"
#include "MyFuncs.h"
#include "Geo/geoFuncs.h"
#include "ScrollsModel.h"
// Structures for the compact validation code
struct chkStructDouble
{
   char str[255];
   double min,max,value;
};
struct chkStructString
{
   char var[255];
   double min,max;
};
// Private function prototypes
int matchStateInputs(struct stateInputVals *stateInputs,struct rowVals *rows, int nC,
     int nR);
int validateStateInputs(struct stateInputVals *stateInputs);
int matchGeoInputs(struct geoInputVals *geoInputs,struct rowVals *rows, int nC, int
    nR);
int validateGeoInputs(struct geoInputVals *geoInputs);
int matchDiscInputs(struct discInputVals *discInputs,struct rowVals *rows, int nC,
    int nR);
int validateDiscInputs(struct discInputVals *discInputs, int nR);
int matchMLInputs(struct MLVals *MLInputs, struct rowVals *rows, int nC, int nR);
int validateMLInputs(struct MLVals *MLInputs, int nR);
int matchFlowInputs(struct FlowVals *FlowInputs, struct rowVals *rows, int nC, int nR
    );
int validateFlowInputs(struct FlowVals *FlowInputs, int nR);
int isWithinRange(double x, double a, double b);
int FlowModel2int(char *string);
int LoadInputs(struct InputVals *Inputs)
{
   // This function reads all the input files located in the folder InputPath (global
        variable)
   char locGeo[400], locDisc[400], locState[400], locML[400], locFlow[400];
   struct rowVals * rows;
   int nC,nR;
   //Build fullpaths to filenames
   strcpy(locGeo,InputPath);
   strcpy(locDisc,InputPath);
   strcpy(locState,InputPath);
   strcpy(locML,InputPath);
   strcpy(locFlow,InputPath);
   strcat(locGeo,"/geoInputs.csv");
   strcat(locState,"/stateInputs.csv");
   strcat(locDisc,"/discInputs.csv");
   strcat(locML,"/MLInputs.csv");
   strcat(locFlow,"/flowsInputs.csv");
   // Build very large empty structures to hold input values
   Inputs->geoInput=(struct geoInputVals *)malloc(500*sizeof(struct geoInputVals));
   Inputs->stateInput=(struct stateInputVals *)malloc(500*sizeof(struct
       stateInputVals));
   Inputs->discInput=(struct discInputVals *)malloc(500*sizeof(struct discInputVals))
       ;
   Inputs->MLInput=(struct MLVals *)malloc(500*sizeof(struct MLVals));
   Inputs->FlowInput=(struct FlowVals *)malloc(500*sizeof(struct FlowVals));
   // Read and validate geometric input values
   rows=LoadFile(locGeo,&nC,&nR);
   matchGeoInputs(Inputs->geoInput,rows,nC,nR);
   validateGeoInputs(Inputs->geoInput);
   free(rows);
   Inputs->Ngeo=nR-1;
   // Read and validate thermodynamic state inputs
```

```
    rows=LoadFile(locState,&nC,&nR);
    matchStateInputs(Inputs->stateInput,rows,nC,nR);
    validateStateInputs(Inputs->stateInput);
    free(rows);
    Inputs->Nstate=nR-1;
    // Read and validate discharge geometry
    rows=LoadFile(locDisc,&nC,&nR);
    matchDiscInputs(Inputs->discInput,rows,nC,nR);
    validateDiscInputs(Inputs->discInput,nR-1); //-1 for the header row
    free(rows);
    Inputs->Ndisc=nR-1;
    // Read and validate mechanical losses and HT terms
    rows=LoadFile(locML,&nC,&nR);
    matchMLInputs(Inputs->MLInput,rows,nC,nR);
    validateMLInputs(Inputs->MLInput,nR-1); //-1 for the header row
    free(rows);
    Inputs->NML=nR-1;
    // Read and validate flow terms
    rows=LoadFile(locFlow,&nC,&nR);
    matchFlowInputs(Inputs->FlowInput,rows,nC,nR);
    validateFlowInputs(Inputs->FlowInput,nR-1); //-1 for the header row
    free(rows);
    Inputs->Nflows=nR-1;
    return 1;
}
void freeInputs(struct InputVals *Inputs)
{
    // Free the Input structures allocated in LoadInputs()
    free(Inputs->geoInput);
    free(Inputs->stateInput);
    free(Inputs->discInput);
    free(Inputs->MLInput);
    free(Inputs->FlowInput);
}
struct rowVals * LoadFile(char *fileName, int *nC, int *nR)
{
    /*
    This function reads in an entire csv-file, parses the file into a row structure
        which
    contains the data for one row of strings and the length of the row, as well as the
         total
    size of the data read in
    */
    int elem,i,j,r;
    struct rowVals * rows;
    int NLine;
    int SkipToNextLine;
    FILE * input;
    long  FileLen; // Length of file
    char *cFile;   // Dynamically allocated buffer (entire file)
    int nLF=0, nCR=0;
    // Set nC and nR to zero in case fileName is actually empty file or the path is
        wrong
    *nC=0; *nR=0;
    // Try to open the file for reading
    input = fopen (fileName , "r");
    if (input==NULL)
    {
        // If it fails to open for reading, print an error and quit
        printf("File [%s] could not be found/opened!\nExiting, bye!...\n\n",fileName);
        exit(-1);
    }
    fseek(input, 0L, SEEK_END); /* Position to end of file */
    FileLen = ftell(input);     /* Get file length */
    rewind(input);              /* Back to start of file */
    /* +1 for '\0' character */
    cFile = calloc(FileLen +1, sizeof(char));
    if(cFile == NULL ) // If it fails to allocate memory
    {
        printf("\nInsufficient memory to read file.\n");
        return 0;
    }
    // Read the entire file into cFile
    fread(cFile, FileLen, 1, input);
    fclose(input); // Close the input
    // This block counts up the number of new line characters,
```

```
// so that the right size block of memory can be allocated
NLine=0;
for (i=FileLen-1;i>=0;i--)
{
   if ( cFile[i]==(char)13)
   {
      // Remove the CR character by bumping all
      // characters forwards
      for (j=i;j<FileLen-1;j++)
      {
         cFile[j]=cFile[j+1];
      }
      FileLen--;
   }
   if ( cFile[i]==(char)10)
      nLF++;
   if ( cFile[i]==(char)13 || cFile[i]==(char)10 )
      NLine++;
}
/*Check whether line endings are ok*/
if (!(nCR==NLine || nLF==NLine))
{
   printf("Number of CR: %d\n Number of LF: %d\nNumber of Lines: %d\n",nCR,nLF,
       NLine);
   printf("Sorry but your input file [%s] does not conform to either Unix\n or Mac
        OS X line ending standards.  Please convert line endings\n to either LF or
        CR and try again\n",fileName);
   exit(-1);
}
rows=(struct rowVals *)malloc((NLine+1)*sizeof(struct rowVals));
//printf_plus("%d Lines Found %d characters",NLine,FileLen);
r=0;
elem=0;
j=0;
SkipToNextLine=0;
for (i=0;i<FileLen;i++)
{
   /* If invalid char (perhaps end of file) end the file loading */
   /*Only check line skipping if you are at the first
   element of the row and not the first row
   This is true if the previous character is an
   end of line character, either LF or CR */
   if (i==0 || ( cFile[i-1]==(char)13 || cFile[i-1]==(char)10 ) )
   {
      /* If the first element is not a '0' or '#' or ',' or '"', process the line,
      otherwise skip it */
      if (cFile[i]=='0' || cFile[i]=='#' || cFile[i]==',' || cFile[i]=='"')
      {
         /*This line of the input file is not activated
         (has a non-one value in the mask column)*/
         SkipToNextLine=1;
         //printf("Skip %c %c %c\n",cFile[i],cFile[i-1],cFile[i-2]);
      }
      else
      {
         SkipToNextLine=0;
         //printf_plus("NoSkip [%d] %c\n",i,cFile[i]);
      }
   }
   if ( cFile[i]==(char)13 || cFile[i]==(char)10 || cFile[i]==(char)23)
   {
      if (SkipToNextLine==0)
      {
         rows[r].strings[elem][j]='\0';
         /*printf("\nrow: %d elem: %d\n",r,elem);
         for (k=0;k<=elem;k++)
         {
            printf("%s,",rows[r].strings[k]);
         }
         printf("\n");*/
         r++;
         *nC=elem+1;
      }
      elem=0;j=0;
      continue;
   }
   if (SkipToNextLine==0)
   {
      if (cFile[i]!=',')
```

```
            {
                rows[r].strings[elem][j]=cFile[i];
                j++;
            }
            else
            {
                /* Terminate previous string with 0 char */
                rows[r].strings[elem][j]='\0';
                elem++;
                j=0;
            }
        }
    }
    free(cFile);
    *nR=r;
    return rows;
}
int matchGeoInputs(struct geoInputVals *geoInputs, struct rowVals *rows, int nC, int
    nR)
{
    int r,i,OK=0;
    /* Loop over the rows that are being used
    Row 0 is the header row which describes the
    variables to be loaded */
    for (r=1;r<nR;r++)
    {
        //Loop over the columns
        for(i=0;i<nC;i++)
        {
            OK=0;
            if (!strcmp(rows[0].strings[i],"rb"))
            { geoInputs[r-1].rb=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"hs"))
            { geoInputs[r-1].hs=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"phi_fi0"))
            { geoInputs[r-1].phi_fi0=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"phi_fis"))
            { geoInputs[r-1].phi_fis=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"phi_fie"))
            { geoInputs[r-1].phi_fie=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"phi_fo0"))
            { geoInputs[r-1].phi_fo0=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"phi_fos"))
            { geoInputs[r-1].phi_fos=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"phi_foe"))
            { geoInputs[r-1].phi_foe=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"phi_oi0"))
            { geoInputs[r-1].phi_oi0=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"phi_ois"))
            { geoInputs[r-1].phi_ois=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"phi_oie"))
            { geoInputs[r-1].phi_oie=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"phi_oo0"))
            { geoInputs[r-1].phi_oo0=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"phi_oos"))
            { geoInputs[r-1].phi_oos=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"phi_ooe"))
            { geoInputs[r-1].phi_ooe=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"A_sa_suction"))
            { geoInputs[r-1].A_sa_suction=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"wall_r"))
            { geoInputs[r-1].wall_r=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"mask"))
            { OK=1;}
            if (OK==0)
            {
                printf("Header term not matched [%s]",rows[0].strings[i]);
                exit(-1);
            }
        }
    }
    printf_plus("%d Geometry Input(s) loaded.....",nR-1); //-1 for header row
```

```
   return 1;
}
int validateGeoInputs(struct geoInputVals *geoInputs)
{
   if (!isWithinRange(geoInputs->phi_fi0,-10,10))
      {printf_plus("\n--------Input File Error------phi_fi0 not valid [%g]...exiting
          ...\n",geoInputs->phi_fi0);
      exit(-1);}
   if (!isWithinRange(geoInputs->phi_fis,-10,10))
      {printf_plus("\n--------Input File Error------phi_fis not valid [%g]...exiting
          ...\n",geoInputs->phi_fis);
      exit(-1);}
   if (!isWithinRange(geoInputs->phi_fie,3.14159,40))
      {printf_plus("\n--------Input File Error------phi_fie not valid [%g]...exiting
          ...\n",geoInputs->phi_fie);
      exit(-1);}
   if (!isWithinRange(geoInputs->phi_fo0,-10,10))
      {printf_plus("\n--------Input File Error------phi_fo0 not valid [%g]...exiting
          ...\n",geoInputs->phi_fo0);
      exit(-1);}
   if (!isWithinRange(geoInputs->phi_fos,-10,10))
      {printf_plus("\n--------Input File Error------phi_fos not valid [%g]...exiting
          ...\n",geoInputs->phi_fos);
      exit(-1);}
   if (!isWithinRange(geoInputs->phi_foe,3.14159,40))
      {printf_plus("\n--------Input File Error------phi_foe not valid [%g]...exiting
          ...\n",geoInputs->phi_foe);
      exit(-1);}
   if (!isWithinRange(geoInputs->phi_oi0,-10,10))
      {printf_plus("\n--------Input File Error------phi_oi0 not valid [%g]...exiting
          ...\n",geoInputs->phi_oi0);
      exit(-1);}
   if (!isWithinRange(geoInputs->phi_ois,-10,10))
      {printf_plus("\n--------Input File Error------phi_ois not valid [%g]...exiting
          ...\n",geoInputs->phi_ois);
      exit(-1);}
   if (!isWithinRange(geoInputs->phi_oie,3.14159,40))
      {printf_plus("\n--------Input File Error------phi_oie not valid [%g]...exiting
          ...\n",geoInputs->phi_oie);
      exit(-1);}
   if (!isWithinRange(geoInputs->phi_oo0,-10,10))
      {printf_plus("\n--------Input File Error------phi_oo0 not valid [%g]...exiting
          ...\n",geoInputs->phi_oo0);
      exit(-1);}
   if (!isWithinRange(geoInputs->phi_oos,-10,10))
      {printf_plus("\n--------Input File Error------phi_oos not valid [%g]...exiting
          ...\n",geoInputs->phi_oos);
      exit(-1);}
   if (!isWithinRange(geoInputs->phi_ooe,3.14159,40))
      {printf_plus("\n--------Input File Error------phi_ooe not valid [%g]...exiting
          ...\n",geoInputs->phi_ooe);
      exit(-1);}
   if ( geoInputs->phi_oos <= geoInputs->phi_ois - PI )
   {printf_plus("Starting angles not valid due to scroll collision... \nCheck phi_os
       and phi_is angles... \nRequirement: phi_os >= phi_is-pi\nExiting...\n");
   exit(-1);}
   printf_plus("Validated.....\n");
   return 1;
}
int matchStateInputs(struct stateInputVals *stateInputs, struct rowVals *rows, int nC
    , int nR)
{
   int r,i,OK=0;
   /* Loop over the rows that are being used
   Row 0 is the header row which describes the
   variables to be loaded */
   for (r=1;r<nR;r++)
   {
      //Loop over the columns
      for(i=0;i<nC;i++)
      {
         OK=0;
         if (!strcmp(rows[0].strings[i],"T_suction"))
         { stateInputs[r-1].T_in=strtod(rows[r].strings[i],NULL); OK=1; }
         if (!strcmp(rows[0].strings[i],"p_suction"))
         { stateInputs[r-1].p_in=strtod(rows[r].strings[i],NULL); OK=1; }
         if (!strcmp(rows[0].strings[i],"p_discharge"))
```

```
            { stateInputs[r-1].p_out=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"xL"))
            { stateInputs[r-1].xL_in=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"T_amb"))
            { stateInputs[r-1].T_amb=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"mdot_exper"))
            { stateInputs[r-1].mdot_exper=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"Power_exper"))
            { stateInputs[r-1].Power_exper=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"Td_exper"))
            { stateInputs[r-1].Td_exper=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"etac_exper"))
            { stateInputs[r-1].etac_exper=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"etav_exper"))
            { stateInputs[r-1].etav_exper=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"omega"))
            { stateInputs[r-1].omega=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"Gas"))
            { strcpy(stateInputs[r-1].Ref,rows[r].strings[i]); OK=1; }
            if (!strcmp(rows[0].strings[i],"Liq"))
            { strcpy(stateInputs[r-1].Liq,rows[r].strings[i]); OK=1; }
            if (!strcmp(rows[0].strings[i],"mask"))
            { OK=1;}
            if (!strcmp(rows[0].strings[i],"z.caseIndex"))
            { OK=1;}
            if (OK==0)
            {
               printf_plus("Header term not matched [%s]",rows[0].strings[i]);
               exit(-1);
            }
         }
      }
      printf_plus("%d State Input(s) Loaded.....",nR-1); //-1 for header row
      return 1;
}
int validateStateInputs(struct stateInputVals *stateInputs)
{
      /* If new fluids are added to the program, additional
      entries need to be added to the list of acceptable
      fluids */
      if (strcmp(stateInputs->Ref,"N2") &&
         strcmp(stateInputs->Ref,"CO2") &&
         strcmp(stateInputs->Ref,"R404a") &&
         strcmp(stateInputs->Ref,"R134a") &&
         strcmp(stateInputs->Ref,"R410A") )
      {
         printf_plus("\n--------Input File Error------Refigerant not valid [%s]...
             exiting...\n",stateInputs->Ref);
         exit(-1);
      }
      if (strcmp(stateInputs->Liq,"Water") &&
          strcmp(stateInputs->Liq,"POE") &&
          strcmp(stateInputs->Liq,"Zerol") )
      {
         printf_plus("\n--------Input File Error------Liq not valid [%s]...exiting...\n"
             ,stateInputs->Liq);
         exit(-1);
      }
      if (stateInputs->T_in<100 || stateInputs->T_in>1000 ) // Units of K
      {
         printf_plus("\n--------Input File Error------Input temperature not valid [%g
             ]...exiting...\n",stateInputs->T_in);
         exit(-1);
      }
      if (stateInputs->p_in<10 || stateInputs->p_in>100000 ) //Units of kPa
      {
         printf_plus("\n--------Input File Error------Input pressure not valid [%g]...
             exiting...\n",stateInputs->p_in);
         exit(-1);
      }
      if (stateInputs->p_out<10 || stateInputs->p_out>100000 ) // Units of kPa
      {
```

```
        printf_plus("\n--------Input File Error------Outlet pressure not valid [%g]...
            exiting...\n",stateInputs->p_out);
        exit(-1);
    }
    if (stateInputs->xL_in<0 || stateInputs->xL_in>1 ) // No units
    {
        printf_plus("\n--------Input File Error------Input Oil Mass Fraction not valid
            [%g]...exiting...\n",stateInputs->xL_in);
        exit(-1);
    }
    if (stateInputs->T_amb<200 || stateInputs->T_amb>400 ) // Units of K
    {
        printf_plus("\n--------Input File Error------Ambient Temperature not valid [%g
            ]...exiting...\n",stateInputs->T_amb);
        exit(-1);
    }
    if (stateInputs->omega<100 || stateInputs->omega>600 ) // Units of rad/s
    {
        printf_plus("\n--------Input File Error------Omega not valid [%g]...exiting...\
            n",stateInputs->omega);
        exit(-1);
    }
    if ( stateInputs->mdot_exper>2 ) //Units of kg/s
    {
        printf_plus("\n--------Input File Error------Experimental mass flow rate not
            valid [%g]...exiting...\n",stateInputs->mdot_exper);
        exit(-1);
    }
    if (stateInputs->Td_exper>500 ) //Units of K
    {
        printf_plus("\n--------Input File Error------Experimental discharge temperature
             not valid [%g]...exiting...\n",stateInputs->Td_exper);
        exit(-1);
    }
    if (stateInputs->Power_exper>20 ) //Units of kW
    {
        printf_plus("\n--------Input File Error------Experimental power not valid [%g
            ]...exiting...\n",stateInputs->Power_exper);
        exit(-1);
    }
    if (stateInputs->etac_exper>1.0 ) //No Units
    {
        printf_plus("\n--------Input File Error------Experimental Adiabatic Efficiency
            not valid [%g]...exiting...\n",stateInputs->etac_exper);
        exit(-1);
    }
    if (stateInputs->etav_exper>1.2 ) //No Units
    {
        printf_plus("\n--------Input File Error------Experimental Volumetric Efficiency
             not valid [%g]...exiting...\n",stateInputs->etav_exper);
        exit(-1);
    }
    printf_plus("Validated.....\n");
    return 1;
}
int matchDiscInputs(struct discInputVals *discInputs, struct rowVals *rows, int nC,
    int nR)
{
    int r,i,OK=0;

    /* Loop over the rows that are being used
    Row 0 is the header row which describes the
    variables to be loaded */
    for (r=1;r<nR;r++)
    {
        //Loop over the columns
        for(i=0;i<nC;i++)
        {
            OK=0;
            if (!strcmp(rows[0].strings[i],"Type"))
            {
                strcpy(discInputs[r-1].Type,rows[r].strings[i]);
                OK=1;
            }
            if (!strcmp(rows[0].strings[i],"xa_arc1"))
            { discInputs[r-1].xa_arc1=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"ya_arc1"))
            { discInputs[r-1].ya_arc1=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"ra_arc1"))
```

```
            { discInputs[r-1].ra_arc1=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"t1_arc1"))
            { discInputs[r-1].t1_arc1=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"t2_arc1"))
            { discInputs[r-1].t2_arc1=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"m_line"))
            { discInputs[r-1].m_line=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"b_line"))
            { discInputs[r-1].b_line=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"t1_line"))
            { discInputs[r-1].t1_line=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"t2_line"))
            { discInputs[r-1].t2_line=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"xa_arc2"))
            { discInputs[r-1].xa_arc2=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"ya_arc2"))
            { discInputs[r-1].ya_arc2=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"ra_arc2"))
            { discInputs[r-1].ra_arc2=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"t1_arc2"))
            { discInputs[r-1].t1_arc2=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"t2_arc2"))
            { discInputs[r-1].t2_arc2=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"disc_x0"))
            { discInputs[r-1].disc_x0=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"disc_y0"))
            { discInputs[r-1].disc_y0=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"disc_R"))
            { discInputs[r-1].disc_R=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"disc_Cd"))
            { discInputs[r-1].disc_Cd=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"mask"))
            { OK=1;}
            //if (!strcmp(rows[0].strings[i],"z.caseIndex"))
            //{ OK=1;}
            if (OK==0)
            {
               printf_plus("Header term not matched [%s]\n\n",rows[0].strings[i]);
               exit(-1);
            }
         }
      }
      printf_plus("%d Discharge Input(s) loaded.....",nR-1); //-1 for header row
      return 1;
}
int validateDiscInputs(struct discInputVals *discInputs, int nR)
{
      int i,r;
      struct chkStructDouble in[2];
      for (r=0;r<nR;r++)
      {
         // Validate the two radii
         in[0].value=discInputs[r].ra_arc1;
         in[0].min=0.0; in[0].max=1.0;
         strcpy(in[0].str,"ra_arc1");
         if (!isWithinRange(discInputs[r].disc_Cd,0.0,1.1))
         {printf_plus("\n--------Input File Error------%s not within [%g,%g] at data row
              %d...exiting...\n","disc_Cd",0,1,r);
         exit(-1);}
         if (!isWithinRange(discInputs[r].disc_R,-1.1,0.1))
         {printf_plus("\n--------Input File Error------%s not within [%g,%g] at data row
              %d...exiting...\n","disc_r",-1.1,0.1,r);
         exit(-1);}
         if (strcmp(discInputs[r].Type,"ArcLineArc")&&strcmp(discInputs[r].Type,"
             ArcLineArc-PMP")&&strcmp(discInputs[r].Type,"2Arc")&&strcmp(discInputs[r].
             Type,"2Arc-PMP"))
         {
```

```
            printf_plus("\n--------Input File Error------\nInvalid type of discharge [%s
                ]\nValid types are \"ArcLineArc\",\"ArcLineArc-PMP\",\"2Arc\",\"2Arc-PMP
                \"",discInputs[r].Type);
            exit(EXIT_FAILURE);
        }
        if (!strcmp(discInputs[r].Type,"ArcLineArc"))
        {
            // Validate the two radii
            in[0].value=discInputs[r].ra_arc1;
            in[0].min=0.0; in[0].max=1.0;
            strcpy(in[0].str,"ra_arc1");
            in[1].value=discInputs[r].ra_arc2;
            in[1].min=0.0; in[1].max=1.0;
            strcpy(in[1].str,"ra_arc2");
            for (i=0;i<2;i++)
            {
                if (!isWithinRange(in[i].value,in[i].min,in[i].max))
                {printf_plus("\n--------Input File Error------%s not within [%g,%g] at
                    data row %d...exiting...\n",in[i].str,in[i].min,in[i].max,r);
                exit(-1);}
            }
        }
        if (!strcmp(discInputs[r].Type,"ArcLineArc-PMP")||!strcmp(discInputs[r].Type,"2
            Arc"))
        {
            // Validate the one radii
            in[0].value=discInputs[r].ra_arc2;
            in[0].min=0.0; in[0].max=1.0;
            strcpy(in[0].str,"ra_arc2");
            for (i=0;i<1;i++)
            {
                if (!isWithinRange(in[i].value,in[i].min,in[i].max))
                {printf_plus("\n--------Input File Error------%s not within [%g,%g] at
                    data row %d...exiting...\n",in[i].str,in[i].min,in[i].max,r);
                exit(-1);}
            }
        }
    }
    printf_plus("Validated.....\n");
    return 1;
}
int matchMLInputs(struct MLVals *MLInputs, struct rowVals *rows, int nC, int nR)
{
    int r,i,OK=0;
    /* Loop over the rows that are being used
    Row 0 is the header row which describes the
    variables to be loaded */
    for (r=1;r<nR;r++)
    {
        //Loop over the columns
        for(i=0;i<nC;i++)
        {
            OK=0;
            if (!strcmp(rows[0].strings[i],"Type"))
            {
                strcpy(MLInputs[r-1].Type,rows[r].strings[i]);
                OK=1;
            }
            if (!strcmp(rows[0].strings[i],"m"))
            { MLInputs[r-1].m=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"b"))
            { MLInputs[r-1].b=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"c"))
            { MLInputs[r-1].c=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"eta_m"))
            { MLInputs[r-1].eta_m=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"UA") || !strcmp(rows[0].strings[i],"UA_amb")
                )
            { MLInputs[r-1].UA_amb=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"etac_guess"))
            { MLInputs[r-1].etac_guess=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"mask"))
            { OK=1;}
            if (OK==0)
```

```
            {
               printf_plus("Header term not matched [%s]\n\n",rows[0].strings[i]);
               exit(-1);
            }
         }
      }
      printf_plus("%d Mechanical Loss Input(s) loaded.....",nR-1); //-1 for header row
      return 1;
}
int validateMLInputs(struct MLVals *MLInputs, int nR)
{
      int i,r;
      struct chkStructDouble in[6];
      for (r=0;r<nR;r++)
      {
         // Fill in temporary value to make it validate eta_m properly
         if (!strcmp(MLInputs[r].Type,"m-b-c"))
            MLInputs[r].eta_m=0.5;
         in[0].value=MLInputs[r].etac_guess;
         in[0].min=0.0; in[0].max=1.0;
         strcpy(in[0].str,"etac_guess");
         in[1].value=MLInputs[r].eta_m;
         in[1].min=0.0; in[1].max=1.0;
         strcpy(in[1].str,"etam");
         in[2].value=MLInputs[r].UA_amb;
         in[2].min=0.0; in[2].max=0.02;
         strcpy(in[2].str,"UA_amb");
         in[3].value=MLInputs[r].c;
         in[3].min=-100.0; in[3].max=100.0;
         strcpy(in[3].str,"ML c");
         in[4].value=MLInputs[r].b;
         in[4].min=-100.0; in[4].max=100.0;
         strcpy(in[4].str,"ML b");
         in[5].value=MLInputs[r].m;
         in[5].min=-100.0; in[5].max=100.0;
         strcpy(in[5].str,"ML m");

         for (i=0;i<6;i++)
         {
            if (!isWithinRange(in[i].value,in[i].min,in[i].max))
            {printf_plus("\n--------Input File Error------\n%s not within [%g,%g]...
                exiting...\n",in[i].str,in[i].min,in[i].max);
            exit(-1);}
         }
      }
      printf_plus("Validated.....\n");
      return 1;
}
int matchFlowInputs(struct FlowVals *FlowInputs, struct rowVals *rows, int nC, int nR
    )
{
      int r,i,OK=0;
      /* Loop over the rows that are being used
      Row 0 is the header row which describes the
      variables to be loaded */
      for (r=1;r<nR;r++)
      {
         //Loop over the columns
         for(i=0;i<nC;i++)
         {
            OK=0;
            if (!strcmp(rows[0].strings[i],"w_ent"))
            { FlowInputs[r-1].w_ent=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"sigma"))
            { FlowInputs[r-1].sigma=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"Z_D_bends"))
            { FlowInputs[r-1].Z_D_bends=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"L_inlet"))
            { FlowInputs[r-1].L_inlet=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"D_inlet"))
            { FlowInputs[r-1].D_inlet=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"L_flank"))
            { FlowInputs[r-1].L_flank=strtod(rows[r].strings[i],NULL); OK=1; }
```

```
            if (!strcmp(rows[0].strings[i],"delta_flank"))
            { FlowInputs[r-1].delta_flank=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"delta_radial"))
            { FlowInputs[r-1].delta_radial=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"phi_flank"))
            { FlowInputs[r-1].phi_flank=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"Cd_inlet"))
            { FlowInputs[r-1].Cd_inlet=strtod(rows[r].strings[i],NULL); OK=1; }
            if (!strcmp(rows[0].strings[i],"flowModel_flank"))
            { FlowInputs[r-1].flowModel_flank=FlowModel2int(rows[r].strings[i]); OK=1; }
            if (!strcmp(rows[0].strings[i],"flowModel_radial"))
            { FlowInputs[r-1].flowModel_radial=FlowModel2int(rows[r].strings[i]); OK=1;
                }
            if (!strcmp(rows[0].strings[i],"flowModel_suction"))
            { FlowInputs[r-1].flowModel_suction=FlowModel2int(rows[r].strings[i]); OK=1;
                 }
            if (!strcmp(rows[0].strings[i],"flowModel_discharge"))
            { FlowInputs[r-1].flowModel_discharge=FlowModel2int(rows[r].strings[i]); OK
                =1; }
            if (!strcmp(rows[0].strings[i],"flowModel_s_sa"))
            { FlowInputs[r-1].flowModel_s_sa=FlowModel2int(rows[r].strings[i]); OK=1; }
            if (!strcmp(rows[0].strings[i],"flowModel_d_dd"))
            { FlowInputs[r-1].flowModel_d_dd=FlowModel2int(rows[r].strings[i]); OK=1; }
            if (!strcmp(rows[0].strings[i],"mask"))
            { OK=1;}
            if (OK==0)
            {
               printf_plus("Header term not matched [%s]\n\n",rows[0].strings[i]);
               exit(-1);
            }
         }
      }
      printf_plus("%d Flow Input(s) loaded.....",nR-1); //-1 for header row
      return 1;
}
int validateFlowInputs(struct FlowVals *FlowInputs, int nR)
{
      int ModelMin=3,ModelMax=8;
      if (!isWithinRange(FlowInputs->Cd_inlet,0.0,1.01))
      {  printf_plus("\n--------Input File Error------%s not within [%g,%g]...exiting
          ...\n","Cd_inlet",0.0,1.01);
         exit(-1);}
      if (FlowInputs->flowModel_suction<ModelMin && FlowInputs->flowModel_suction>
          ModelMax)
         {printf_plus("Suction flow model not valid [%d]...exiting...\n",FlowInputs->
             flowModel_suction);
         exit(-1);}
      if (FlowInputs->flowModel_discharge<ModelMin && FlowInputs->flowModel_discharge>
          ModelMax)
         {printf_plus("Discharge flow model not valid [%d]...exiting...\n",FlowInputs->
             flowModel_discharge);
         exit(-1);}
      if (FlowInputs->flowModel_s_sa<ModelMin && FlowInputs->flowModel_s_sa>ModelMax)
         {printf_plus("s-sa flow model not valid [%d]...exiting...\n",FlowInputs->
             flowModel_s_sa);
         exit(-1);}
      if (FlowInputs->flowModel_d_dd<ModelMin && FlowInputs->flowModel_d_dd>ModelMax)
         {printf_plus("d-dd flow model not valid [%d]...exiting...\n",FlowInputs->
             flowModel_d_dd);
         exit(-1);}
      if (FlowInputs->flowModel_flank<ModelMin && FlowInputs->flowModel_flank>ModelMax)
         {printf_plus("Flank flow model not valid [%d]...exiting...\n",FlowInputs->
             flowModel_flank);
         exit(-1);}
      if (FlowInputs->flowModel_radial<ModelMin && FlowInputs->flowModel_radial>ModelMax
          )
         {printf_plus("Radial flow model not valid [%d]...exiting...\n",FlowInputs->
             flowModel_radial);
         exit(-1);}
      printf_plus("Validated.....\n");
      return 1;
}
int isWithinRange(double x, double a, double b)
```

```
{
   return (x>=a && x<=b);
}
int FlowModel2int(char *string)
{
   // Takes in a string representation of the flow model, and converts to integer (
       using macro definitions)
   if (!strcmp(string,"DRY_GAS_FLANK_FRICTIONAL_MODEL"))
      return DRY_GAS_FLANK_FRICTIONAL_MODEL;
   else if (!strcmp(string,"DRY_GAS_RADIAL_FRICTIONAL_MODEL"))
      return DRY_GAS_RADIAL_FRICTIONAL_MODEL;
   else if (!strcmp(string,"DRY_GAS_FLANK_FLANK_MODEL"))
      return DRY_GAS_FLANK_FLANK_MODEL;
   else if (!strcmp(string,"TWO_PHASE_NOZZLE"))
      return TWO_PHASE_NOZZLE;
   else if (!strcmp(string,"TEE_FLOW_MODEL"))
      return TEE_FLOW_MODEL;
   else if (!strcmp(string,"BENDS_MODEL"))
      return BENDS_MODEL;
   else if (!strcmp(string,"CORRECTED_RADIAL_NOZZLE"))
      return CORRECTED_RADIAL_NOZZLE;
   else if (!strcmp(string,"CORRECTED_FLANK_NOZZLE"))
      return CORRECTED_FLANK_NOZZLE;
   else if (!strcmp(string,"LIQUID_RADIAL_FRICTIONAL_MODEL"))
      return LIQUID_RADIAL_FRICTIONAL_MODEL;
   else if (!strcmp(string,"LIQUID_FLANK_FRICTIONAL_MODEL"))
      return LIQUID_FLANK_FRICTIONAL_MODEL;
   else
      return -1;
}
```

SaveOutputs.h

```
#ifndef SAVEOUTPUTS
#define SAVEOUTPUTS
   void saveOutputs(struct scrollVals *scroll);
   char batchPath[2000];
   char runPath[2000];
   char logPath[2000];
   char YMD[12];
   char baseOutputFolder[1000];
   void BuildBatchPath();
   int BatchBuilt;
#endif
```

SaveOutputs.c

```
#ifndef __GNUC__
#define _CRT_SECURE_NO_WARNINGS
#define _CRTDBG_MAP_ALLOC
#include <stdlib.h>
#include <crtdbg.h>
#else
#include <stdlib.h>
#endif
#include <stdio.h>
#include <string.h>
#include "math.h"
#include "time.h"
#include "StructsMacros.h"
#include "MyFuncs.h"
#include "Geo/geoFuncs.h"
#include "ScrollsModel.h"
#include "Props/FloodProp.h"
#include "MassFlow.h"
#include "HeatTransfer.h"
#include "SaveOutputs.h"
#include "LoadInputs.h"
#include "CompressorModel.h"
void BuildBatchPath()
{
   /*
   This block of code counts the number of existing folders
   in the day's folder, and then makes the batch folder be the next
   numbered batch
   Also adds a batch description to the batch folder name
   */
   char slash[2];
```

```
    char str[1000];
    char dayPath[500];
    time_t curtime;
    struct tm *loctime;
    struct rowVals *rows;
    int nC;
    if (!BatchBuilt)
    {
        if (runNumber==1)
        {
            #if defined(__WIN32__) || defined(__WIN64__) || defined(_MSC_VER)
                strcpy(slash,"\\");
                //strcpy(dayPath,"..\\..\\ScrollModel_Results");
            #else
                strcpy(slash,"/");
                //strcpy(dayPath,"../../ScrollModel_Results");
            #endif
            strcpy(dayPath,baseOutputFolder);
            makeDir(dayPath);
            strcpy(logPath,dayPath);
            strcat(logPath,slash);
            strcat(logPath,"BatchLog");
            makeDir(logPath);
            // This block of code composes the
            // Year-Month-Day form for making the output folder
            curtime=time(NULL);
            loctime=localtime(&curtime);
            strftime(YMD,11,"%Y_%m_%d",loctime);
            strcat(dayPath,slash);
            strcat(dayPath,YMD);
            makeDir(dayPath);
            // Take a system dir call and store the call to a text file in the day
                folder
            #if defined(__WIN32__) || defined(__WIN64__)
            // option /AD is only directories
            // option /B is just the bare output (just the names of the directories, one
                per line)
            sprintf(str,"dir %s%sBatch* /AD /B > %s%sdir.txt",dayPath,slash,dayPath,
                slash); // Warning: Windows-specific syntax
            system(str);
            #else
            sprintf(str,"ls -d %s/Batch*> %s/dir.txt",dayPath,dayPath);
            //printf("%s",str);
            system(str);
            #endif
            /*Read the txt file back in and count the number of rows into
            the variable batchNumber (each row is one directory) */
            sprintf(str,"%s%sdir.txt",dayPath,slash);
            rows=LoadFile(str,&nC,&batchNumber);
            free(rows);
            #if defined(__WIN32__) || defined(__WIN64__)
            sprintf(str,"erase %s\\dir.txt",dayPath); // Warning: Windows-specific
                syntax
            system(str);
            #else
            //printf("rm %s/dir.txt",dayPath);
            sprintf(str,"rm %s/dir.txt",dayPath); // Warning: Linux-specific syntax
            system(str);
            #endif
            //It is assumed that every folder is a batch - if not, no harm, the batch
                folder number will just be higher
            //batchNumber--; // Subtract 1 for the current batch folder (the . in the
                directory call)
            sprintf(str,"%sBatch %04d (%s)",slash,batchNumber+1,batchDescription);
            strcpy(batchPath,dayPath); // copy dayPath to batchPath
            strcat(batchPath,str); //Tack on batch number and description
            makeDir(batchPath);
        }
        BatchBuilt=1;
    }
}
void saveOutputs(struct scrollVals *scroll)
{
    int i;
    FILE *fp, *fpdisc;
    // Hierarchical names for folders
    char PythonPath[200],strPython[200];
```

```
char fileName[100];
struct rowVals *rows;
int nC,nR,elem;
char outString[200000],headString[200000];
char str[1000],strMakePlots[1000];
BuildBatchPath();
// This block gives a four digit numbered folder for each run
strcpy(runPath,batchPath);
sprintf(str,"/Run %04d",runNumber);
strcat(runPath,str);
makeDir(runPath);
strcpy(str,InputPath);
strcat(str,"/templatedOutput.csv");
rows=LoadFile(str,&nC,&nR);
//printf_plus("%d %d %s\n",nC,nR,rows[0].strings[2]);
headString[0]='\0';
outString[0]='\0';
elem=0;
strcat(headString,"Run,");
sprintf(str,"%d,",runNumber);
strcat(outString,str);
for (i=0;i<nC;i++)
{
   if (!strcmp(rows[elem].strings[i],"MainResults"))
   {
      /*
      Cannot mix experimental points with experimental data and
      points without experimental data in a given batch.
      Should not generally be a problem
      */
      if (scroll->Exper.mdot>0) //There are experimental values
      {
      strcat(headString,"|||Main|||,mdot_model [kg/s],mdot_exper[kg/s],mdot_err[%]
          ");
      sprintf(str,"||||||,%0.12f,%0.12f,%0.12f",scroll->massFlow.mdot_tot,scroll->
          Exper.mdot,(scroll->massFlow.mdot_tot-scroll->Exper.mdot)/scroll->Exper.
          mdot*100.0);
      strcat(outString,str);
      strcat(headString,",P_model [kW],P_exper [kW],P_err [%]");
      sprintf(str,",%0.12f,%0.12f,%0.12f",scroll->PowerEff.P_shaft,scroll->Exper.
          P_shaft,(scroll->PowerEff.P_shaft-scroll->Exper.P_shaft)/scroll->Exper.
          P_shaft*100.0);
      strcat(outString,str);
      strcat(headString,",Td_model [K],Td_exper [K],Td_err [K]");
      sprintf(str,",%0.12f,%0.12f,%0.12f",scroll->T[Idischarge],scroll->Exper.T_d,
          scroll->T[Idischarge]-scroll->Exper.T_d);
      strcat(outString,str);
      strcat(headString,",eta_c");
      sprintf(str,",%0.12f",scroll->PowerEff.eta_c);
      strcat(outString,str);
      strcat(headString,",eta_v");
      sprintf(str,",%0.12f",scroll->PowerEff.eta_v);
      strcat(outString,str);
      }
      else
      {
      strcat(headString,"mdot [kg/s]");
      sprintf(str,"%0.12f",scroll->massFlow.mdot_tot);
      strcat(outString,str);
      strcat(headString,",P_shaft [kW]");
      sprintf(str,",%0.12f",scroll->PowerEff.P_shaft);
      strcat(outString,str);
      strcat(headString,",Td [K]");
      sprintf(str,",%0.12f",scroll->T[Idischarge]);
      strcat(outString,str);
      strcat(headString,",eta_c [-]");
      sprintf(str,",%0.12f",scroll->PowerEff.eta_c);
      strcat(outString,str);
      strcat(headString,",eta_v [-]");
      sprintf(str,",%0.12f",scroll->PowerEff.eta_v);
      strcat(outString,str);
      }
   }
```

```
if (!strcmp(rows[elem].strings[i],"Inputs"))
{
   strcat(headString,"|||Inputs|||,Ref,Liq,T_in [K],p_in [kPa],p_out [kPa], xL
       [-],omega[rad/s],T_amb [K],p_ratio [-]");
   sprintf(str,"|||||,%s,%s,%0.12f,%0.12f,%0.12f,%0.12f,%0.12f,%0.12f,%0.12f",
      scroll->Ref, scroll->Liq, scroll->Inputs.T_in, scroll->Inputs.p_in,
      scroll->Inputs.p_out, scroll->Inputs.xL_in, scroll->Inputs.omega,
      scroll->Inputs.T_amb,scroll->Inputs.p_out/scroll->Inputs.p_in);
   strcat(outString,str);
}
if (!strcmp(rows[elem].strings[i],"Geometry"))
{
   strcat(headString,"|||Geometry|||,rb [m],hs [m],phi_i0 [rad],phi_is [rad],
       phi_ie [rad],phi_o0 [rad],phi_os [rad],phi_oe [rad],theta_d [rad],Vdisp
       [m^3],Vratio [-]");
   sprintf(str,"||||||,%0.12f,%0.12f,%0.12f,%0.12f,%0.12f,%0.12f,%0.12f,%0.12f
       ,%0.12f,%0.5e,%0.12f",
      scroll->geo.rb,scroll->geo.hs,scroll->geo.phi.phi_fi0,scroll->geo.phi.
          phi_fis,
      scroll->geo.phi.phi_fie,scroll->geo.phi.phi_fo0,scroll->geo.phi.phi_fos,
      scroll->geo.phi.phi_foe,theta_d(&(scroll->geo)),Vdisp(&(scroll->geo)),
          Vratio(&(scroll->geo)));
   strcat(outString,str);
}
if (!strcmp(rows[elem].strings[i],"PowerEff"))
{
   strcat(headString,"|||Powers|||,P_shaft [kW], P_gas [kW], P_ML [kW], eta_m
       [-]");
   sprintf(str,"||||||,%0.12f,%0.12f,%0.12f,%0.12f",
      scroll->PowerEff.P_shaft,scroll->PowerEff.P_gas,scroll->PowerEff.P_ML,
          scroll->PowerEff.eta_m);
   strcat(outString,str);
}
if (!strcmp(rows[elem].strings[i],"HT"))
{
   strcat(headString,"|||HT|||,Q_scrolls_gas [kW], Q_scroll_inlet [kW],
       Q_scroll_outlet [kW], Q_scroll_plenum [kW],Q_scroll_amb [kW],T_scroll[K
       ],UA_amb [kW/m^2K]");
   sprintf(str,"|||||,%0.8f,%0.8f,%0.8f,%0.8f,%0.8f,%0.8f,%0.8f",
      scroll->HT.Q_scroll_gas,scroll->HT.Q_scroll_inlet,scroll->HT.
          Q_scroll_outlet,
      scroll->HT.Q_scroll_plenum,scroll->HT.Q_scroll_amb,scroll->HT.T_scroll,
          scroll->ML.UA_amb );
   strcat(outString,str);
}
if (!strcmp(rows[elem].strings[i],"ML"))
{
   strcat(headString,"|||ML|||,P_ML [kW],m [kW], b [-],c [-]");
   sprintf(str,"|||||,%0.8f,%0.8f,%0.8f,%0.8f",
      scroll->PowerEff.P_ML,scroll->ML.m,scroll->ML.b,scroll->ML.c);
   strcat(outString,str);
   if (scroll->Exper.mdot>0) //There are experimental values
   {
      strcat(headString,",P_MLexper [kW]");
      sprintf(str,",%0.8f",scroll->Exper.P_shaft-scroll->PowerEff.P_gas);
      strcat(outString,str);
   }
}
if (!strcmp(rows[elem].strings[i],"FlowInputs"))
{
   strcat(headString,"|||Flow|||,A_sa_suct [m^2],Cd_inlet [-],w_ent [-],sigma
       [-], Z_D [-],L_inlet [m],D_inlet [m],L_flank [m], phi_flank [rad],
       delta_flank [um], delta_radial [um],FM_flank, FM_radial,FM_suction,
       FM_discharge,FM_s_sa,FM_d_dd");
   sprintf(str,"|||||,%0.8f,%0.4f,%0.8f,%0.8f,%0.8f,%0.8f,%0.8f,%0.8f,%0.8f
       ,%0.8f,%0.8f,%d,%d,%d,%d,%d,%d",
      scroll->geo.suct.A_sa_suction,scroll->massFlow.Inputs.Cd_inlet,
      scroll->massFlow.Inputs.w_ent,scroll->massFlow.Inputs.sigma,scroll->
          massFlow.Inputs.Z_D_bends,
      scroll->massFlow.Inputs.L_inlet,scroll->massFlow.Inputs.D_inlet,scroll->
          massFlow.Inputs.L_flank,
      scroll->massFlow.Inputs.phi_flank,scroll->massFlow.Inputs.delta_flank*1.0
          e6,
      scroll->massFlow.Inputs.delta_radial*1.0e6,scroll->geo.flowModels.flank,
      scroll->geo.flowModels.radial,scroll->geo.flowModels.suction, scroll->geo
          .flowModels.discharge,
      scroll->geo.flowModels.s_sa, scroll->geo.flowModels.d_dd);
```

```
        strcat(outString,str);
    }
    if (!strcmp(rows[elem].strings[i],"Disc"))
    {
        strcat(headString,"|||Disc|||,Type,ra_arc1[m],ra_arc2 [m],x_port [m],y_port
            [m],r_port [m],Cd_disc [-]");
        sprintf(str,"|||||,%s,%0.8f,%0.8f,%0.8f,%0.8f,%0.8f,%0.4f",
           scroll->geo.disc.Type,scroll->geo.disc.ra_arc1,scroll->geo.disc.ra_arc2,
               scroll->geo.disc.x0,
           scroll->geo.disc.y0,scroll->geo.disc.R,scroll->geo.disc.Cd);
        strcat(outString,str);
    }
    if (!strcmp(rows[elem].strings[i],"Debug") || !strcmp(rows[elem].strings[i],"
     Debugs"))
    {
        strcat(headString,"|||Debug|||,mdot_error [kg/s], wrap_eror [%], HTerror [kW
            ], Tderror [K],Ntheta [-], RunTime [min]");
        sprintf(str,"|||||,%0.8f,%0.8f,%0.8f,%0.8f,%d,%0.8f",
           scroll->Debug.mdot_error_abs, scroll->Debug.wrap_error_rel,
           scroll->Debug.LumpHT_error_abs,scroll->Debug.Td_error_abs,scroll->Debug.
               Ntheta,scroll->Debug.ElapsedTime);
        strcat(outString,str);
    }
    if (!strcmp(rows[elem].strings[i],"Losses"))
    {
        strcat(headString,"|||Losses|||,W_adiabatic [kW], Leakage-Flank [kW],
            Leakage-Radial [kW], Suction [kW], Discharge [kW],Mechanical [kW]");
        sprintf(str,"|||||,%0.8f,%0.8f,%0.8f,%0.8f,%0.8f,%0.8f",
           scroll->Losses.Wdot_adiabatic,scroll->Losses.leakage_flank,scroll->Losses
               .leakage_radial,
           scroll->Losses.suction,scroll->Losses.discharge,scroll->Losses.mechanical
               );
        strcat(outString,str);
    }
    if (!strcmp(rows[elem].strings[i],"Forces"))
    {
        strcat(headString,"|||Forces|||,Fx_mean [kN], Fy_mean [kN], Fz_mean [kN],
            Mx_mean [kN], My_mean [kN], Mz_mean [kN], Torque_mean [kNm],
            Fradial_mean [kN],M_pin_mean [kNm]");
        sprintf(str,"|||||,%0.8f,%0.8f,%0.8f,%0.8f,%0.8f,%0.8f,%0.8f,%0.8f,%0.8f",
           scroll->Forces.Fx_mean,scroll->Forces.Fy_mean,scroll->Forces.Fz_mean,
           scroll->Forces.Mx_mean,scroll->Forces.My_mean,scroll->Forces.Mz_mean,
           scroll->Forces.tau_mean,scroll->Forces.Frad_mean,scroll->Forces.MO_mean);
        strcat(outString,str);
    }
    strcat(headString,",");
    strcat(outString,",");
}
// Output to Batch Results File
sprintf(fileName,"%s/%s",batchPath,"Results.csv");
fp=fopen(fileName,"a");
while(fp==NULL)
{
printf_plus("%s cannot be accessed\n",fileName);
printf_plus("\a");
fp=fopen(fileName,"a");
}
if (runNumber==1)
   fprintf(fp,"%s\n",headString);
fprintf(fp,"%s\n",outString);
fclose(fp);
// Output to BatchLog File
sprintf(str,"%s/%s",logPath,"BatchLog.csv");
fp=fopen(str,"a");
while(fp==NULL)
{
printf_plus("%s cannot be accessed\n",str);
printf_plus("\a");
fp=fopen(str,"a");
}
if (runNumber==1)
{
   fprintf(fp,"Date: %s : Batch %04d %s\n",YMD,batchNumber+1,batchDescription);
   fprintf(fp,"%s\n",headString);
}
fprintf(fp,"%s\n",outString);
```

```
fclose(fp);
free(rows);
sprintf(fileName,"%s/%s",runPath,"theta.csv");
Matrix2csv(fileName,scroll->theta,1,scroll->Ntheta);
sprintf(fileName,"%s/%s",runPath,"V.csv");
Matrix2csv(fileName,scroll->V,NCV,scroll->Ntheta);
sprintf(fileName,"%s/%s",runPath,"dV.csv");
Matrix2csv(fileName,scroll->dV,NCV,scroll->Ntheta);
sprintf(fileName,"%s/%s",runPath,"flowVec.csv");
flowVec2csv(fileName,scroll,scroll->Ntheta);
sprintf(fileName,"%s/%s",runPath,"T.csv");
Matrix2csv(fileName,scroll->T,NCV,scroll->Ntheta);
sprintf(fileName,"%s/%s",runPath,"p.csv");
Matrix2csv(fileName,scroll->p,NCV,scroll->Ntheta);
sprintf(fileName,"%s/%s",runPath,"xL.csv");
Matrix2csv(fileName,scroll->xL,NCV,scroll->Ntheta);
sprintf(fileName,"%s/%s",runPath,"Fx.csv");
Matrix2csv(fileName,scroll->Forces.Fx,NCV,scroll->Ntheta);
sprintf(fileName,"%s/%s",runPath,"Fy.csv");
Matrix2csv(fileName,scroll->Forces.Fy,NCV,scroll->Ntheta);
sprintf(fileName,"%s/%s",runPath,"Fz.csv");
Matrix2csv(fileName,scroll->Forces.Fz,NCV,scroll->Ntheta);
sprintf(fileName,"%s/%s",runPath,"MO.csv");
Matrix2csv(fileName,scroll->Forces.MO,NCV,scroll->Ntheta);
sprintf(fileName,"%s/%s",runPath,"Q.csv");
Matrix2csv(fileName,scroll->Q,NCV,scroll->Ntheta);
sprintf(fileName,"%s/%s",runPath,"hc.csv");
Matrix2csv(fileName,scroll->HT.hc,NCV,scroll->Ntheta);
sprintf(fileName,"%s/%s",runPath,"A_wall_i.csv");
Matrix2csv(fileName,scroll->HT.A_wall_i,NCV,scroll->Ntheta);
sprintf(fileName,"%s/%s",runPath,"A_wall_o.csv");
Matrix2csv(fileName,scroll->HT.A_wall_o,NCV,scroll->Ntheta);
sprintf(fileName,"%s/%s",runPath,"Tm_wall_i.csv");
Matrix2csv(fileName,scroll->HT.Tm_wall_i,NCV,scroll->Ntheta);
sprintf(fileName,"%s/%s",runPath,"Tm_wall_o.csv");
Matrix2csv(fileName,scroll->HT.Tm_wall_o,NCV,scroll->Ntheta);
sprintf(fileName,"%s/%s",runPath,"Tm_plate.csv");
Matrix2csv(fileName,scroll->HT.Tm_plate,NCV,scroll->Ntheta);
sprintf(fileName,"%s/%s",runPath,"error.csv");
Matrix2csv(fileName,scroll->error,3*NCV,scroll->Ntheta);
sprintf(fileName,"%s/%s",runPath,"Adisc.csv");
fpdisc=fopen(fileName,"w");
for (i=0;i<NTHETA_ADISC;i++)
{
   fprintf(fpdisc,"%0.12f,%0.12f\n",scroll->geo.disc.thetaAdisc[i],scroll->geo.
       disc.Adisc[i]);
}
fclose(fpdisc);
sprintf(fileName,"%s/%s",runPath,"Indices.csv");
fp=fopen(fileName,"w");
fprintf(fp,"Isa,%d\n",Isa);
fprintf(fp,"Is1,%d\n",Is1);
fprintf(fp,"Is2,%d\n",Is2);
fprintf(fp,"Id1,%d\n",Id1);
fprintf(fp,"Id2,%d\n",Id2);
fprintf(fp,"Idd,%d\n",Idd);
fprintf(fp,"Iddd,%d\n",Iddd);
fprintf(fp,"Isuction,%d\n",Isuction);
fprintf(fp,"Idischarge,%d\n",Idischarge);
fprintf(fp,"Ic1");
for (i=0;i<nC_Max(&(scroll->geo));i++)
   fprintf(fp,",%d",Ic1[i]);
fprintf(fp,"\n");
fprintf(fp,"Ic2");
for (i=0;i<nC_Max(&(scroll->geo));i++)
   fprintf(fp,",%d",Ic2[i]);
fprintf(fp,"\n");
fclose(fp);
#if defined(__WIN32__) || defined(__WIN64__) || defined(_MSC_VER)
```

```
    strcpy(PythonPath,"c:\\Python26\\python.exe");
    sprintf(strPython,"dir /B %s",PythonPath);
    sprintf(strMakePlots,"dir /B makePlots.py");
    #else
    strcpy(PythonPath,"/opt/python/2.6/bin/python");
    sprintf(strPython,"ls %s",PythonPath);
    sprintf(strMakePlots,"ls makePlots.py");
    #endif
    if (!system(strPython))
    {
       if (!system(strMakePlots))
       {
          // swap backslashes and forward slashes for the call to Python
          for (i=0;i<strlen(runPath);i++)
             if (runPath[i]=='\\') runPath[i]='/';
          //Compose terminal call
          sprintf(str,"%s %s \"%s\"",PythonPath,"makePlots.py",runPath);
          system(str);
       }
       else
       {
          printf_plus("makePlots.py not found in executable folder...");
       }
    }
    else
    {
       printf_plus("Python not found in %s, please update path to Python in
           saveOutputs.c",PythonPath);
    }
    #if defined(__WIN32__) || defined(__WIN64__) || defined(_MSC_VER)
       sprintf(str,"7zip %s/DATA.zip %s/*.csv",runPath,runPath);
       system(str);
       sprintf(str,"erase %s/*.csv",runPath);
       system(str);
    #else
       // move to the run folder, zip all csv into a bz2 archive, remove CSV files,
           make a code folder, copy .c or .h files into code folder, zip into a bz2
           archive, remove all the c and h files from the code folder, move bz2 here,
           delete code folder
       sprintf(str,"cd \"%s\" && tar -cjf DATA.tar.bz2 *.csv&& rm *.csv && mkdir code
           && cp ~/SYNC/*.[ch] code && cp ~/SYNC/Geo/*.[ch] code && cp ~/SYNC/Props
           /*.[ch] code && tar -cjf code/CODE.tar.bz2 code/*.[ch] && rm code/*.[ch] &&
            mv code/CODE.tar.bz2 . && rmdir code",runPath);
       system(str);
    #endif
}
```

geoFuncs.h

```
//File: geoFuncs.h
#include "../StructsMacros.h"
#ifndef GEO_FUNCS
#define GEO_FUNCS
    // ****************************************************************
    //                          Constants/Macros
    // ****************************************************************
    #define PI 3.141592653589793
    #define TWO_PHASE_NOZZLE 3
    #define DRY_GAS_RADIAL_FRICTIONAL_MODEL 4
    #define DRY_GAS_FLANK_FRICTIONAL_MODEL 5
    #define TEE_FLOW_MODEL 6
    #define BENDS_MODEL 7
    #define DRY_GAS_FLANK_FLANK_MODEL 8
    #define CORRECTED_RADIAL_NOZZLE 9
    #define CORRECTED_FLANK_NOZZLE 10
    #define LIQUID_RADIAL_FRICTIONAL_MODEL 11
    #define LIQUID_FLANK_FRICTIONAL_MODEL 12
    int Isa,Is1,Is2,*Ic1,*Ic2,Id1,Id2,Idd,Iddd,Isuction,Idischarge,*Iinjection;
    int NCV;
    // ****************************************************************
    //                      Function Prototypes
    // ****************************************************************
    void coords_inv(double phi, double theta,struct geoVals * geo, char *code, double
        *x, double *y);
    void coords_norm(double phi, char *code, double *nx, double *ny);
    void sortAnglesCW(double *t1, double *t2);
    void sortAnglesCCW(double *t1, double *t2);
```

```
    void printPhi(struct geoVals * geo);
    // double conj(double theta, int k,char *code, struct geoVals * geo);
    double Vs_Wang(struct geoVals *geo ,double theta);
    double phi_s_sa(struct geoVals * geo,double theta);
    double phi_d_dd(struct geoVals * geo,double theta);
    double polyArea(double *x,double*y,int N);
    void CVCentroid(struct geoVals *geo, double theta, double phiOuter1, double
        phiOuter2,char* codeOuter,double phiInner1,double phiInner2,char *codeInner,
        int isSuction,/*in out*/ double *cx,double *cy);
    double Vdisp(struct geoVals *geo);
    double Vratio(struct geoVals *geo);
    int nC(struct geoVals *geo,double theta);
    int nC_Max(struct geoVals *geo);
    double theta_d(struct geoVals *geo);
    struct flowVecVals buildFlowVec(struct geoVals *geo,double theta, int useDDD,int
        LeftDischarge);
    void freeFlowVec(struct flowVecVals *flowVec);
    void LoadCVIndices(struct geoVals * geo);
    void addRadialLeak(struct flowVecVals *flowVec,struct geoVals *geo, int CV1, int
        CV2, double phi_max,double phi_min, int radialFlowModel, int *count);
    void addFlankLeak(struct flowVecVals *flowVec,struct geoVals *geo, int CV1, int
        CV2, int flankFlowModel, int *count);
    void addPrimaryFlow(struct flowVecVals *flowVec,struct geoVals *geo, int CV1, int
        CV2, double theta,char *path,int FlowModel, int *count);
    void GeometryModel(struct geoVals *geo,double theta, int Istep, int useDDD, int
        LeftDischarge, double *V, double *dV, struct flowVecVals * flowVec);
    double A_disc(struct geoVals *geo,double theta);
    double interpVec(double *t, double *A, double t_goal, int N);
    void cleanUpGeo();
    double Vc(struct geoVals *geo,double theta,int alpha);
    // New functions
    void Vs1_calcs(struct geoVals *geo,double theta,double*V, double *dV,double *cx,
        double*cy);
    void Vc1_calcs(struct geoVals *geo,double theta,int alpha,double*V, double *dV,
        double *cx,double*cy);
    void Vd1_calcs(struct geoVals *geo,double theta,double*V, double *dV,double *cx,
        double*cy);
    void Vdd_calcs(struct geoVals *geo,double theta,double*V, double *dV,double *cx,
        double*cy);
    void setDiscGeo(struct geoVals *geo,char *Type);
#endif
```

geoFuncs.c

```
//geoFuncs.c
// ------------------------------
// To make sense of all the geometry equations, refer to
//  Bell, I., "Theoretical and Experimental Analysis of
//  Liquid Flooded Compression in Scroll Compressors", PhD Thesis,
//  Purdue University, May 2011
#ifndef __GNUC__
#define _CRT_SECURE_NO_WARNINGS
#define _CRTDBG_MAP_ALLOC
#include <stdlib.h>
#include <crtdbg.h>
#else
#include <stdlib.h>
#endif
#include <stdio.h>
#include <string.h>
#include "gpc.h"
#include "math.h"
#include "../StructsMacros.h"
#include "geoFuncs.h"
#include "../MyFuncs.h"
//************************************************
//       Angles and Basic Geometry Functions
//************************************************
void coords_inv(double phi, double theta,struct geoVals * geo, char *code, double *x,
     double *y)
{
    // Function calculates the involute coordinates of a point given the involute
        angle and the scroll geometry
    // outputs are x and y, all other parameters are inputs
    //Equivalent MATLAB call:
```

```
    //function [x, y]=coords_inv(phi,theta,geo,code)
    int done=0;
    double theta_m;
    double rb,phi_fi0,phi_fo0,phi_fie,phi_oie,ro;
    struct phiVals *phis;
    phis=&(geo->phi);
    rb=geo->rb;
    ro=geo->ro;
    phi_fie=phis->phi_fie;
    phi_oie=phis->phi_oie;

    if (strcmp(code,"fi")==0)
    {
       phi_fi0=phis->phi_fi0;
       //case 'fi'  %fixed inner involute
       *x=rb*(cos(phi)+(phi-phi_fi0)*sin(phi));
       *y=rb*(sin(phi)-(phi-phi_fi0)*cos(phi));
       done=1;
    }
    if (strcmp(code,"fo")==0)
    {
       //case 'fo' %fixed outer involute
       phi_fo0=phis->phi_fo0;
         *x=rb*(cos(phi)+(phi-phi_fo0)*sin(phi));
       *y=rb*(sin(phi)-(phi-phi_fo0)*cos(phi));
       done=1;
    }
     if (strcmp(code,"oi")==0)
     {
        //case 'oi' %Orbiting inner involute
       phi_fi0=phis->phi_fi0;
         theta_m=-PI/2.0+phi_fie-theta;
         *x=-rb*(cos(phi)+(phi-phi_fi0)*sin(phi))+ro*cos(theta_m);
         *y=-rb*(sin(phi)-(phi-phi_fi0)*cos(phi))+ro*sin(theta_m);
         done=1;
     }
     if (strcmp(code,"oo")==0)
     {
        //case 'oo' %Orbiting outer involute
       phi_fo0=phis->phi_fo0;
         theta_m=-PI/2.0+phi_fie-theta;
         *x=-rb*(cos(phi)+(phi-phi_fo0)*sin(phi))+ro*cos(theta_m);
         *y=-rb*(sin(phi)-(phi-phi_fo0)*cos(phi))+ro*sin(theta_m);
         done=1;
     }
     if (done==0)
     {
        // Didn't succeed
        printf_plus("Uh oh. Case incorrect in coords_inv");
     }
}
void coords_norm(double phi, char *code, double *nx, double *ny)
{
    int done=0;
    if (strcmp(code,"fi")==0)
    {
       //case 'fi'  %fixed inner involute
       *nx=sin(phi);
       *ny=-cos(phi);
       done=1;
    }
    if (strcmp(code,"fo")==0)
    {
       //case 'fo'  %fixed outer involute
       *nx=-sin(phi);
       *ny=cos(phi);
       done=1;
    }
    if (strcmp(code,"oi")==0)
    {
       //case 'oi'  %orbiting inner involute
       *nx=-sin(phi);
       *ny=cos(phi);
       done=1;
    }
    if (strcmp(code,"oo")==0)
    {
       //case 'oo'  %orbiting outer involute
       *nx=sin(phi);
       *ny=-cos(phi);
       done=1;
```

```
    }
    if (done==0)
     {
        // Didn't succeed
        printf_plus("Uh oh. Case incorrect in coords_inv");
     }
}
void LoadCVIndices(struct geoVals * geo)
{
    int N,i;
    Isa=0;
    Is1=1;
    Id1=2;
    Is2=3;
    Id2=4;
    Idd=5;
    Iddd=6;
    Isuction=7;
    Idischarge=8;
    /*Iinjection=(int)calloc(1,sizeof(int));
    Iinjection[0]=9;*/
    /*  Start compression chambers at +9
     *  If more indices are needed, (for injection ports or other reasons)
     *  the 9 can be increased, or the new chambers can be added
     *  after the compression chambers indices
     */
    if (!Ic1) // If Ic1 is still undefined
    {
        N=nC_Max(geo);
        Ic1=(int *)calloc(N,sizeof(int));
        Ic2=(int *)calloc(N,sizeof(int));
        for (i=0;i<N;i++)
        {
            Ic1[i]=9+i;
            Ic2[i]=9+i+N;
        }
        NCV=Ic2[i-1]+1;//+1 since need to count 0-th index
    }
}
void printPhi(struct geoVals * geo)
{
    //Formatted printout of the involute angles
    printf ("phi_fi0:%f radians\n",geo->phi.phi_fi0);
    printf ("phi_fis:%f radians\n",geo->phi.phi_fis);
    printf ("phi_fie:%f radians\n",geo->phi.phi_fie);
    printf ("phi_fo0:%f radians\n",geo->phi.phi_fo0);
    printf ("phi_fos:%f radians\n",geo->phi.phi_fos);
    printf ("phi_foe:%f radians\n",geo->phi.phi_foe);
    printf ("phi_oi0:%f radians\n",geo->phi.phi_oi0);
    printf ("phi_ois:%f radians\n",geo->phi.phi_ois);
    printf ("phi_oie:%f radians\n",geo->phi.phi_oie);
    printf ("phi_oo0:%f radians\n",geo->phi.phi_oo0);
    printf ("phi_oos:%f radians\n",geo->phi.phi_oos);
    printf ("phi_ooe:%f radians\n",geo->phi.phi_ooe);
}

void sortAnglesCW(double *t1, double *t2)
{
    double temp;
    //Sort angles so that t2>t1 in a clockwise sense
    // idea from http://stackoverflow.com/questions/242404/sort-four-points-in-
        clockwise-order
    // more description: http://softsurfer.com/Archive/algorithm_0101/algorithm_0101.
        htm
    /* If the signed area of the triangle formed between the points on a unit circle
    and the originis positive, the angles are sorted counterclockwise. Otherwise, the
        angles
    are sorted in a counter-clockwise manner.  Here we want the angles to be sorted CW
        , so
    if area is negative, swap angles
    Area obtained from the cross product of a vector from origin
    to 1 and a vector to point 2, so use right hand rule to get
    sign of cross product with unit length
    */
    if (cos(*t1)*sin(*t2)-cos(*t2)*sin(*t1)>0)
    {
        //Swap angles
        temp=*t1;
        *t1=*t2;
```

```
      *t2=temp;
   }
   while (*t1 > *t2)
   {
      // Make t2 bigger than t1
      *t2+=2*PI;
   }
}
void sortAnglesCCW(double *t1, double *t2)
{
   double temp;
   //Sort angles so that t2>t1 in a counter-clockwise sense
   // idea from http://stackoverflow.com/questions/242404/sort-four-points-in-
       clockwise-order
   // more description: http://softsurfer.com/Archive/algorithm_0101/algorithm_0101.
       htm
   /* If the signed area of the triangle formed between the points on a unit circle
       with angles t1 and t2
   and the origin is positive, the angles are sorted counterclockwise. Otherwise, the
       angles
   are sorted in a counter-clockwise manner.  Here we want the angles to be sorted
       CCW, so
   if area is negative, swap angles
   Area obtained from the cross product of a vector from origin
   to 1 and a vector to point 2, so use right hand rule to get
   sign of cross product with unit length
   */
   if (cos(*t1)*sin(*t2)-cos(*t2)*sin(*t1)<0)
   {
      //Swap angles
      temp=*t1;
      *t1=*t2;
      *t2=temp;
   }
   while (*t1 > *t2)
   {
      // Make t2 bigger than t1
      *t2+=2*PI;
   }
}
double phi_s_sa(struct geoVals * geo,double theta)
{
   return (PI-(geo->phi.phi_fi0)+(geo->phi.phi_fo0))/((geo->phi.phi_fie)-(geo->phi.
       phi_oo0)-PI)*sin(theta)+(geo->phi.phi_fie)-PI;
}
double theta_d(struct geoVals *geo)
{
   return geo->phi.phi_fie-geo->phi.phi_fos-2*PI*nC_Max(geo)-PI;
}
double phi_d_dd(struct geoVals * geo,double theta)
{
   int iter;
   double phi_os,phi_o0,phi_ie,phi_i0,change,eps,f;
   double x1,x2,x3,y1,y2,phi,alpha;
   phi_os=geo->phi.phi_fos;
   phi_o0=geo->phi.phi_fo0;
   phi_ie=geo->phi.phi_fie;
   phi_i0=geo->phi.phi_fi0;
   alpha=PI-phi_i0+phi_o0;
   //Use secant method to calculate the involute angle at break
   eps=1e-8;
   change=999;
   iter=1;
   while ((iter<=3 || change>eps) && iter<100)
   {
      if (iter==1){x1=geo->phi.phi_fis; phi=x1;}
      if (iter==2){x2=geo->phi.phi_fis+0.1; phi=x2;}
      if (iter>2) {phi=x2;}
         f=1+cos(phi-phi_os)-(phi_os-phi_o0)*sin(phi-phi_os)+alpha*sin(phi-phi_ie+
             theta);
      if (iter==1){y1=f;}
      if (iter==2){y2=f;}
      if (iter>2)
      {
         y2=f;
         x3=x2-y2/(y2-y1)*(x2-x1);
         change=fabs(y2/(y2-y1)*(x2-x1));
```

```
            y1=y2; x1=x2; x2=x3;
        }
        iter=iter+1;
        // If the value is still less than the starting angle
        // after 20 iterations
        if (iter>20 && x3<geo->phi.phi_fis)
        {
            return geo->phi.phi_fis;
        }
    }
    if (x3>geo->phi.phi_fis)
        return x3;
    else
    {
        return geo->phi.phi_fis;
    }
    //return phi_os+PI;
}
double polyArea(double *x,double*y,int N)
{
    // Takes in a non-closed set of curves which form a polygon and
    // determines the area between the curves
    // Technically this formula gives a negative area if the
    // points are oriented clockwise, but the absolute value is
    // taken to eliminate the problem
    double sum=0;
    int i=0;
    for (i=0;i<N-2;i++)
    {
        sum+=x[i]*y[i+1]-x[i+1]*y[i];
    }
    //Close the two polygons
    sum+=x[N-1]*y[0]-x[0]*y[N-1];
    return fabs(sum/2.0);
}
int nC(struct geoVals *geo,double theta)
{
    return (int)((geo->phi.phi_fie-theta-geo->phi.phi_fos-PI)/(2.0*PI));
}
int nC_Max(struct geoVals *geo)
{
    return (int)((geo->phi.phi_fie-geo->phi.phi_fos-PI)/(2.0*PI));
}
// ************************************************
//                  Volumes and Centroids
// ************************************************
double Vs_Wang(struct geoVals *geo ,double theta)
{
    double h,a,r,phi_e,alpha_i,alpha_o;
    h=geo->hs;
    a=geo->rb;
    r=geo->ro;
    phi_e=geo->phi.phi_fie;
    alpha_i=geo->phi.phi_fi0;
    alpha_o=geo->phi.phi_fo0;
    return h/2.0*a*r*(2.0*theta*phi_e-theta*theta-theta*(alpha_i+alpha_o+PI)+2.0*(1.0-
        cos(theta))-2.0*(phi_e-PI)*sin(theta)-PI/4*sin(2*theta));
}
double fxA_FixedInvolute(double rb, double phi,double phi0)
{
    // Anti-derivative term into which is substituted for integrations between the
        origin and an involute of the fixed scroll
     return rb*rb*rb/3.0*(4.0*(powInt(phi-phi0,2)-2.0)*sin(phi)+(phi0-phi)*(powInt(phi
         -phi0,2)-8.0)*cos(phi));
}
double fyA_FixedInvolute(double rb, double phi, double phi0)
{
    // Anti-derivative term into which is substituted for integrations between the
        origin and an involute of the fixed scroll
     return rb*rb*rb/3.0*((phi0-phi)*(powInt(phi-phi0,2)-8.0)*sin(phi)-4.0*(powInt(phi
         -phi0,2)-2.0)*cos(phi));
}
void Vs1_calcs(struct geoVals *geo,double theta,double *Vs,double *dVs,double *cx,
    double *cy)
{
    double B,h,ro,rb,phi_e,phi_o0,phi_i0,phi_ie,b,D,B_prime, V0,dV0,cx_0,cy_0,VIa,dVIa
        ,cx_Ia,cy_Ia,VIb,dVIb,cx_Ib,cy_Ib,VIc,dVIc,cx_Ic,cy_Ic,cx_I,cy_I;
    h=geo->hs;
```

```
    ro=geo->ro;
    rb=geo->rb;
    phi_e=geo->phi.phi_fie;
    phi_ie=geo->phi.phi_fie;
    phi_o0=geo->phi.phi_oo0;
    phi_i0=geo->phi.phi_oi0;
    b=(-phi_o0+phi_e-PI);
     D=ro/rb*((phi_i0-phi_e)*sin(theta)-cos(theta)+1)/(phi_e-phi_i0);
     B=1.0/2.0*(sqrt(b*b-4.0*D)-b);
     B_prime=-ro/rb*(sin(theta)+(phi_i0-phi_ie)*cos(theta))/((phi_e-phi_i0)*sqrt(b*b
         -4*D));
     V0=h*rb*rb/6.0*(powInt(phi_e-phi_i0,3)-powInt(phi_e-theta-phi_i0,3));
     dV0=h*rb*rb/2.0*(powInt(phi_e-theta-phi_i0,2));
     cx_0=h/V0*(fxA_FixedInvolute(rb,phi_ie,phi_i0)-fxA_FixedInvolute(rb,phi_ie-theta,
         phi_i0));
     cy_0=h/V0*(fyA_FixedInvolute(rb,phi_ie,phi_i0)-fyA_FixedInvolute(rb,phi_ie-theta,
         phi_i0));
     VIa=h*rb*rb/6.0*(powInt(phi_e-PI+B-phi_o0,3)-powInt(phi_e-PI-theta-phi_o0,3));
     dVIa=h*rb*rb/2.0*(powInt(phi_e-PI+B-phi_o0,2)*B_prime+powInt(phi_e-PI-theta-
         phi_o0,2));
     cx_Ia=h/VIa*(fxA_FixedInvolute(rb,phi_ie-PI+B,phi_o0)-fxA_FixedInvolute(rb,phi_ie
         -PI-theta,phi_o0));
     cy_Ia=h/VIa*(fyA_FixedInvolute(rb,phi_ie-PI+B,phi_o0)-fyA_FixedInvolute(rb,phi_ie
         -PI-theta,phi_o0));
     VIb=h*rb*ro/2.0*((B-phi_o0+phi_e-PI)*sin(B+theta)+cos(B+theta));
     dVIb=h*rb*ro*(B_prime+1)/2.0*((phi_e-PI+B-phi_o0)*cos(B+theta)-sin(B+theta));
     cx_Ib=1.0/3.0*(-rb*(B-phi_o0+phi_e-PI)*sin(B+phi_e)-rb*cos(B+phi_e)-ro*sin(theta-
         phi_e));
     cy_Ib=1.0/3.0*(-rb*sin(B+phi_e)+rb*(B-phi_o0+phi_e-PI)*cos(B+phi_e)-ro*cos(theta-
         phi_e));
     VIc=h*rb*ro/2.0;
     dVIc=0;
     cx_Ic=1.0/3.0*(rb*(-theta-phi_o0+phi_e-PI)*sin(theta-phi_e)-ro*sin(theta-phi_e)-
         rb*cos(theta-phi_e));
     cy_Ic=1.0/3.0*(rb*sin(theta-phi_e)+rb*(-theta-phi_o0+phi_e-PI)*cos(theta-phi_e)-
         ro*cos(theta-phi_e));
     cx_I=-(cx_Ia*VIa+cx_Ib*VIb-cx_Ic*VIc)/(VIa+VIb-VIc)+ro*cos(phi_ie-PI/2.0-theta);
     cy_I=-(cy_Ia*VIa+cy_Ib*VIb-cy_Ic*VIc)/(VIa+VIb-VIc)+ro*sin(phi_ie-PI/2.0-theta);
     *Vs=V0-(VIa+VIb-VIc)+1e-9;
     *dVs=dV0-(dVIa+dVIb-dVIc);
     *cx=(cx_0*V0-cx_I*(VIa+VIb-VIc))/ *Vs;
     *cy=(cy_0*V0-cy_I*(VIa+VIb-VIc))/ *Vs;
}
void Vc1_calcs(struct geoVals *geo,double theta,int alpha,double *V,double *dV,double
      *cx,double *cy)
{
   double h,ro,rb,phi_ie,phi_o0,phi_i0,psi;
   h=geo->hs;
   ro=geo->ro;
   rb=geo->rb;
   phi_ie=geo->phi.phi_fie;
   phi_o0=geo->phi.phi_oo0;
   phi_i0=geo->phi.phi_oi0;
   *V=-PI*h*rb*ro*(2*theta+4*alpha*PI-2*phi_ie-PI+phi_i0+phi_o0);
    *dV=-2.0*PI*h*rb*ro;
    psi=rb/3.0*(3.0*theta*theta+6.0*phi_o0*theta+3.0*phi_o0*phi_o0+PI*PI-15.0+(theta+
        phi_o0)*(12.0*PI*alpha-6.0*phi_ie)+3.0*phi_ie*phi_ie+12.0*PI*alpha*(PI*alpha-
        phi_ie))/(2.0*theta+phi_o0-2.0*phi_ie+phi_i0+4.0*PI*alpha-PI);
    *cx=-2.0*rb*cos(theta-phi_ie)-psi*sin(theta-phi_ie);
    *cy=+2.0*rb*sin(theta-phi_ie)-psi*cos(theta-phi_ie);
}
void Vd1_calcs(struct geoVals *geo,double theta,double *V,double *dV,double *cx,
    double *cy)
{
   double h,ro,rb,phi_ie,phi_o0,phi_i0,phi_is,phi_os,V0,dV0,cx_0,cy_0,VIa,dVIa,cx_Ia,
       cy_Ia,VIb,dVIb,cx_Ib,cy_Ib,VIc,dVIc,cx_Ic,cy_Ic,VId,dVId,cx_Id,cy_Id,VI,dVI,
       cx_I,cy_I;
   h=geo->hs;
   ro=geo->ro;
   rb=geo->rb;
   phi_ie=geo->phi.phi_fie;
   phi_o0=geo->phi.phi_oo0;
   phi_i0=geo->phi.phi_oi0;
   phi_is=geo->phi.phi_fis;
   phi_os=geo->phi.phi_fos;
```

```
    VO=h*rb*rb/6.0*(powInt(phi_ie-theta-2.0*PI*nC(geo,theta)-phi_i0,3)-powInt(phi_os+
        PI-phi_i0,3));
    dVO=-h*rb*rb/2.0*(powInt(phi_ie-theta-2.0*PI*nC(geo,theta)-phi_i0,2));
    cx_O=h/VO*(fxA_FixedInvolute(rb,phi_ie-theta-2.0*PI*nC(geo,theta),phi_i0)-
        fxA_FixedInvolute(rb,phi_os+PI,phi_i0));
    cy_O=h/VO*(fyA_FixedInvolute(rb,phi_ie-theta-2.0*PI*nC(geo,theta),phi_i0)-
        fyA_FixedInvolute(rb,phi_os+PI,phi_i0));
    VIa=h*rb*rb/6.0*(powInt(phi_ie-theta-2.0*PI*nC(geo,theta)-PI-phi_o0,3)-powInt(
        phi_os-phi_o0,3));
    dVIa=-h*rb*rb/2.0*(powInt(phi_ie-theta-2.0*PI*nC(geo,theta)-PI-phi_o0,2));
    cx_Ia=h/VIa*(fxA_FixedInvolute(rb,phi_ie-theta-2.0*PI*nC(geo,theta)-PI,phi_o0)-
        fxA_FixedInvolute(rb,phi_os,phi_o0));
    cy_Ia=h/VIa*(fyA_FixedInvolute(rb,phi_ie-theta-2.0*PI*nC(geo,theta)-PI,phi_o0)-
        fyA_FixedInvolute(rb,phi_os,phi_o0));
    VIb=h*rb*ro/2.0*((phi_os-phi_o0)*sin(theta+phi_os-phi_ie)+cos(theta+phi_os-phi_ie
        ));
    dVIb=h*rb*ro/2.0*((phi_os-phi_o0)*cos(theta+phi_os-phi_ie)-sin(theta+phi_os-
        phi_ie));
    cx_Ib=1.0/3.0*(-ro*sin(theta-phi_ie)+rb*(phi_os-phi_o0)*sin(phi_os)+rb*cos(phi_os
        ));
    cy_Ib=1.0/3.0*(-ro*cos(theta-phi_ie)-rb*(phi_os-phi_o0)*cos(phi_os)+rb*sin(phi_os
        ));
    VIc=h*rb*ro/2.0;
    dVIc=0;
    cx_Ic=1.0/3.0*((rb*(-theta+phi_ie-phi_o0-2*PI*nC(geo,theta)-PI)-ro)*sin(theta-
        phi_ie)-rb*cos(theta-phi_ie));
    cy_Ic=1.0/3.0*((rb*(-theta+phi_ie-phi_o0-2*PI*nC(geo,theta)-PI)-ro)*cos(theta-
        phi_ie)+rb*sin(theta-phi_ie));
   VId= h*rb*ro/2.0*((phi_os-phi_i0+PI)*sin(theta+phi_os-phi_ie)+cos(theta+phi_os-
       phi_ie)+1);
    dVId=h*rb*ro/2.0*((phi_os-phi_i0+PI)*cos(theta+phi_os-phi_ie)-sin(theta+phi_os-
        phi_ie));
    cx_Id=(rb*(2*phi_os-phi_o0-phi_i0+PI)*sin(phi_os)-2*(ro*sin(theta-phi_ie)-rb*cos(
        phi_os)))/3.0;
    cy_Id=(-2*(ro*cos(theta-phi_ie)-rb*sin(phi_os))-rb*(2*phi_os-phi_o0-phi_i0+PI)*
        cos(phi_os))/3.0;
    VI=VIa+VIb+VIc+VId;
    dVI=dVIa+dVIb+dVIc+dVId;
    cx_I=-(cx_Ia*VIa+cx_Ib*VIb+cx_Ic*VIc+cx_Id*VId)/VI+ro*cos(phi_ie-PI/2.0-theta);
    cy_I=-(cy_Ia*VIa+cy_Ib*VIb+cy_Ic*VIc+cy_Id*VId)/VI+ro*sin(phi_ie-PI/2.0-theta);
    *V=VO-VI;
    *dV=dVO-dVI;
    *cx=(cx_O*VO-cx_I*VI)/ *V;
    *cy=(cy_O*VO-cy_I*VI)/ *V;
}
void Vdd_calcs(struct geoVals *geo,double theta,double *V,double *dV,double *cx,
    double *cy)
{
   double hs,ro,rb,phi_ie,phi_o0,phi_i0,phi_is,phi_os,VOa,dVOa,VOb,dVOb,VOc,dVOc,
      VIa,dVIa,VIb,dVIb,xa1,ya1,ra1,ta1_2,ta1_1,xa2,ya2,ra2,ta2_1,ta2_2,m_line,
      b_line,t1_line,t2_line,xoos,yoos,x1l,y1l,x2l,ml,om;
   hs=geo->hs;
   ro=geo->ro;
   rb=geo->rb;
   phi_ie=geo->phi.phi_fie;
   phi_o0=geo->phi.phi_oo0;
   phi_i0=geo->phi.phi_oi0;
   phi_is=geo->phi.phi_fis;
   phi_os=geo->phi.phi_fos;
   xa1=geo->disc.xa_arc1;
    ya1=geo->disc.ya_arc1;
    ra1=geo->disc.ra_arc1;
    ta1_1=geo->disc.t1_arc1; //TODO: fix angle definitions
    ta1_2=geo->disc.t2_arc1;
    xa2=geo->disc.xa_arc2;
    ya2=geo->disc.ya_arc2;
    ra2=geo->disc.ra_arc2;
    ta2_2=geo->disc.t2_arc2;
    ta2_1=geo->disc.t1_arc2;
    m_line=geo->disc.m_line;
    b_line=geo->disc.b_line;
    t1_line=geo->disc.t1_line;
    t2_line=geo->disc.t2_line;
   om=phi_ie-PI/2.0-theta;
    coords_inv(phi_os,  theta, geo,"oo",&xoos,&yoos);
    //###################### Oa portion ######################
```

```
    VOa=hs*((-(ra1*(cos(ta1_2)*(ya1-yoos)-sin(ta1_2)*(xa1-xoos)-ra1*ta1_2))/2)-(-(ra1
        *(cos(ta1_1)*(ya1-yoos)-sin(ta1_1)*(xa1-xoos)-ra1*ta1_1))/2));
    dVOa=-hs*ra1*ro/2.0*((sin(om)*sin(ta1_2)+cos(om)*cos(ta1_2))-(sin(om)*sin(ta1_1)+
        cos(om)*cos(ta1_1)));
    //###################### Ob portion ######################
    x1l=t1_line;
    y1l=m_line*t1_line+b_line;
    VOb=hs/2.0*((ro*xoos-ro*x1l)*sin(om)-(ro*cos(om)-2.0*x1l)*yoos+y1l*(ro*cos(om)
        -2.0*xoos));
    dVOb=ro*hs/2.0*(ro-yoos*sin(om)-xoos*cos(om)-y1l*sin(om)-x1l*cos(om));
   //###################### Oc portion ######################
   VOc=rb*hs/6*(
          3*ro*(phi_os-phi_i0+PI)*sin(theta+phi_os-phi_ie)
         +3*ro*cos(theta+phi_os-phi_ie)
         +3*ro*(phi_is-phi_i0)*sin(theta+phi_is-phi_ie)
         +3*ro*cos(theta+phi_is-phi_ie)
         +3*rb*((phi_is-phi_i0)*(phi_os-phi_o0)+1)*sin(phi_os-phi_is)
         -3*rb*(phi_os-phi_o0-phi_is+phi_i0)*cos(phi_os-phi_is)
         +rb*(pow(phi_os+PI-phi_i0,3)-pow(phi_is-phi_i0,3))+3*ro);
    dVOc=rb*hs*ro/2*(
         (phi_os-phi_i0+PI)*cos(theta+phi_os-phi_ie)
         -sin(theta+phi_os-phi_ie)
         +(phi_is-phi_i0)*cos(theta+phi_is-phi_ie)
         -sin(theta+phi_is-phi_ie));
    //###################### Ia portion ######################
    VIa=hs*ra2/2.0*(xa2*(sin(ta2_2)-sin(ta2_1))
                 -ya2*(cos(ta2_2)-cos(ta2_1))
                 -rb*(sin(ta2_2-phi_os)-sin(ta2_1-phi_os))
        -rb*(phi_os-phi_o0)*(cos(ta2_2-phi_os)-cos(ta2_1-phi_os))
                 +ra2*(ta2_2-ta2_1)  );
    dVIa=0.0;
    //###################### Ib portion ######################
    x1l=t1_line;
    x2l=t2_line;
    y1l=m_line*t1_line+b_line;
    ml=m_line;
    VIb=-hs*(x2l-x1l)/2.0*(rb*ml*(cos(phi_os)+(phi_os-phi_o0)*sin(phi_os))+b_line-rb
        *(sin(phi_os)-(phi_os-phi_o0)*cos(phi_os)));
    dVIb=0;
    *cx=ro*cos(om)/2.0;
    *cy=ro*sin(om)/2.0;
    *V =2.0*( VOa +VOb +VOc -VIa -VIb);
    *dV=2.0*(dVOa+dVOb+dVOc-dVIa-dVIb);
}
double dVs(struct geoVals *geo,double theta)
{
   double h,a,r,phi_e,alpha_i,alpha_o;
   h=geo->hs;
   a=geo->rb;
   r=geo->ro;
   phi_e=geo->phi.phi_fie;
   alpha_i=geo->phi.phi_fi0;
   alpha_o=geo->phi.phi_fo0;
   return h/2.0*a*r*(2.0*phi_e-2.0*theta-(alpha_i+alpha_o+PI)+2.0*sin(theta)-2.0*(
       phi_e-PI)*cos(theta)-PI/2.0*cos(2.0*theta));
}
double Vc(struct geoVals *geo,double theta,int alpha)
{
   // Convenience function return the volume directly
   double V,dV,cx,cy;
   Vc1_calcs(geo,theta,alpha,&V,&dV,&cx,&cy);
   return V;
}
double Vdisp(struct geoVals *geo)
{
   return -2.0*PI*(geo->hs)*(geo->rb)*(geo->ro)*(3.0*PI-2.0*geo->phi.phi_fie+(geo->
       phi.phi_fi0+geo->phi.phi_fo0));
}
double Vratio(struct geoVals *geo)
{
   return (3.0*PI-2.0*geo->phi.phi_fie+(geo->phi.phi_fi0+geo->phi.phi_fo0))/(-2.0*geo
       ->phi.phi_fos-3.0*PI+geo->phi.phi_fi0+geo->phi.phi_fo0);
}
// This is the new method, but it doesn't work very well, so sticking
// with the old method for now
```

```
        //double Vsa_integrate(struct geoVals *geo,double theta)
        //{
        // double phi1, phi2,b,D,B,B_prime,rb,ro,phi_0;
        // double phi_ie,phi_o0,phi_i0,phi_is,phi_os,V_Isa;
        // phi_ie=geo->phi.phi_fie;
        // phi_o0=geo->phi.phi_oo0;
        // phi_i0=geo->phi.phi_oi0;
        // phi_is=geo->phi.phi_fis;
        // phi_os=geo->phi.phi_fos;
        // ro=geo->ro;
        // rb=geo->rb;
        // phi_0=geo->phi.phi_fo0;
        //
        // b=(-phi_o0+phi_ie-PI);
        //    D=ro/rb*((phi_i0-phi_ie)*sin(theta)-cos(theta)+1)/(phi_ie-phi_i0);
        //    B=1.0/2.0*(sqrt(b*b-4.0*D)-b);
        //    B_prime=-ro/rb*(sin(theta)+(phi_i0-phi_ie)*cos(theta))/((phi_ie-phi_i0)*
            sqrt(b*b-4*D));
        // V_Isa=(geo->hs)*rb*rb/6.0*(powInt(geo->phi.phi_foe-phi_0,3)-powInt(phi_ie-PI
            +B-phi_0,3));
        // //Volume of SA chamber is equal to area of empty walls minus the outer
            volume of the scroll set
        // return (geo->hs)*PI*powInt(geo->wall.r,2)-2*V_Isa;
        //}
        //double dVsa_integrate(struct geoVals *geo,double theta)
        //{
        // double b,D,B,B_prime,rb,ro,phi_0;
        // double phi_ie,phi_o0,phi_i0,phi_is,phi_os;
        // phi_ie=geo->phi.phi_fie;
        // phi_o0=geo->phi.phi_oo0;
        // phi_i0=geo->phi.phi_oi0;
        // phi_is=geo->phi.phi_fis;
        // phi_os=geo->phi.phi_fos;
        // ro=geo->ro;
        // rb=geo->rb;
        // phi_0=geo->phi.phi_fo0;
        //
        // b=(-phi_o0+phi_ie-PI);
        //    D=ro/rb*((phi_i0-phi_ie)*sin(theta)-cos(theta)+1)/(phi_ie-phi_i0);
        //    B=1.0/2.0*(sqrt(b*b-4.0*D)-b);
        //    B_prime=-ro/rb*(sin(theta)+(phi_i0-phi_ie)*cos(theta))/((phi_ie-phi_i0)*
            sqrt(b*b-4*D));
        //
        // return (geo->hs)*rb*rb*powInt(phi_ie-PI+B-phi_0,2)*B_prime;
        //}
double Vsa_integrate(struct geoVals *geo,double theta)
{
   double phi1, phi2,phi_0;
   double xos,yos,rb,A2,A1;

   rb=geo->rb;
   phi_0=geo->phi.phi_fo0;

   // For the fixed scroll part
   coords_inv(phi_s_sa(geo,theta),theta,geo,"oo",&xos,&yos);
   phi2=geo->phi.phi_foe;
   phi1=phi_s_sa(geo,theta);
   A2=rb/2.0*(rb*powInt(phi2-phi_0,3)/3.0+cos(phi2)*(yos+(phi2-phi_0)*xos)-sin(phi2)
       *(xos-(phi2-phi_0)*yos));
   A1=rb/2.0*(rb*powInt(phi1-phi_0,3)/3.0+cos(phi1)*(yos+(phi1-phi_0)*xos)-sin(phi1)
       *(xos-(phi1-phi_0)*yos));

   //Volume of SA chamber is equal to area of empty walls minus the outer volume of
       the scroll set
   return (geo->hs)*( PI*powInt(geo->wall.r,2)-2*(A2-A1) );
}
double dVsa_integrate(struct geoVals *geo,double theta)
{
   return (Vsa_integrate(geo,theta+1e-5)-Vsa_integrate(geo,theta))/1e-5;
}

//************************************************************
//                      LEAKAGE FUNCTIONS
//************************************************************
void addRadialLeak(struct flowVecVals *flowVec,struct geoVals *geo, int CV1, int CV2,
     double phi_max,double phi_min, int radialFlowModel, int *count)
{
   flowVec->CV1[*count]=CV1;
   flowVec->CV2[*count]=CV2;
```

```
    flowVec->A[*count]=(geo->delta_radial)*(geo->rb)*(1.0/2.0*(phi_max*phi_max-phi_min
        *phi_min)-(geo->phi.phi_fi0)*(phi_max-phi_min));
    flowVec->flowModel[*count]=radialFlowModel;
    *count=*count+1;
}
void addFlankLeak(struct flowVecVals *flowVec,struct geoVals *geo, int CV1, int CV2,
     int flankFlowModel, int *count)
{
    flowVec->CV1[*count]=CV1;
    flowVec->CV2[*count]=CV2;
    flowVec->A[*count]=(geo->delta_flank)*(geo->hs);
    flowVec->flowModel[*count]=flankFlowModel;
    *count=*count+1;
}
void addPrimaryFlow(struct flowVecVals *flowVec,struct geoVals *geo, int CV1, int CV2
     , double theta,char *path,int FlowModel, int *count)
{
    double x_fie,y_fie,x_oob,y_oob,x_fis,y_fis,x_oos,y_oos;
    flowVec->CV1[*count]=CV1;
    flowVec->CV2[*count]=CV2;
    if (!strcmp(path,"s-sa"))
    {
        coords_inv(geo->phi.phi_fie,theta,geo,"fi",&x_fie,&y_fie);
        coords_inv(phi_s_sa(geo,theta),theta,geo,"oo",&x_oob,&y_oob);
        flowVec->A[*count]=(geo->hs)*sqrt((x_fie-x_oob)*(x_fie-x_oob)+(y_fie-y_oob)*(
            y_fie-y_oob));
    }
    if (!strcmp(path,"discharge"))
    {
        /* Interpolate the pre-calculated discharge port blockage using Lagrange
            quadratic interpolation */
        flowVec->A[*count]=geo->disc.Cd*interpVec(&(geo->disc.thetaAdisc),&(geo->disc.
            Adisc),theta,NTHETA_ADISC);
    }
    if (!strcmp(path,"d-dd"))
    {
        coords_inv(phi_d_dd(geo,theta),theta,geo,"fi",&x_fis,&y_fis);
        coords_inv(geo->phi.phi_oos,theta,geo,"oo",&x_oos,&y_oos);
        flowVec->A[*count]=(geo->hs)*sqrt((x_fis-x_oos)*(x_fis-x_oos)+(y_fis-y_oos)*(
            y_fis-y_oos));
        if ((flowVec->A[*count])>1000)
        {
            printf_plus("Error in d-dd area %g>1000\n",flowVec->A[*count]);
            printf_plus("phi_d_dd(geo,theta): %g phi_d_dd(geo,theta-0.001): %g\n",
                phi_d_dd(geo,theta),phi_d_dd(geo,theta-0.001));
        }
    }
    if (!strcmp(path,"suction"))
    {
        flowVec->A[*count]=geo->suct.A_sa_suction;
    }
    flowVec->flowModel[*count]=FlowModel;
    *count=*count+1;
}
struct flowVecVals buildFlowVec(struct geoVals *geo,double theta,int useDDD,int
     LeftDischarge)
{
    int Nmax=100;
    struct flowVecVals flowVec;
    int i, count=0, radialFlowModel,flankFlowModel,s_saFlowModel,suctionFlowModel,
        dischargeFlowModel,d_ddFlowModel,alpha,Nc;
    double phi_ie;

    LoadCVIndices(geo);
    phi_ie=(geo->phi.phi_fie);
    Nc=nC(geo,theta);
    //Initialize structure with all zeros
    flowVec.CV1       =(int *)calloc(Nmax,sizeof(int));
    flowVec.CV2       =(int *)calloc(Nmax,sizeof(int));
    flowVec.CVup      =(int *)calloc(Nmax,sizeof(int));
    flowVec.A         =(double *)calloc(Nmax,sizeof(double));
    flowVec.flowModel =(int *)calloc(Nmax,sizeof(int));
    flowVec.mdot      =(double *)calloc(Nmax,sizeof(double));
    flowVec.mdot_L    =(double *)calloc(Nmax,sizeof(double));
    flowVec.h_up      =(double *)calloc(Nmax,sizeof(double));
    flowVec.h_down    =(double *)calloc(Nmax,sizeof(double));
    flowVec.T_up      =(double *)calloc(Nmax,sizeof(double));
```

```
flowVec.T_down    =(double *)calloc(Nmax,sizeof(double));
flowVec.p_up      =(double *)calloc(Nmax,sizeof(double));
flowVec.p_down    =(double *)calloc(Nmax,sizeof(double));
flowVec.xL        =(double *)calloc(Nmax,sizeof(double));
flowVec.Ed        =(double *)calloc(Nmax,sizeof(double));
flowVec.Re        =(double *)calloc(Nmax,sizeof(double));
flowVec.Ma        =(double *)calloc(Nmax,sizeof(double));
//Load flow model values
radialFlowModel=geo->flowModels.radial;
flankFlowModel=geo->flowModels.flank;
s_saFlowModel=geo->flowModels.s_sa;
suctionFlowModel=geo->flowModels.suction;
d_ddFlowModel=geo->flowModels.d_dd;
dischargeFlowModel=geo->flowModels.discharge;

if ((phi_ie-theta > phi_s_sa(geo,theta)) && Nc==0)
{
   printf_plus("Volume ratio too small and cannot solve");
}
//Fill radial values
if (Nc>0)
{
   //Handle the suction area angles
   if (phi_ie-theta > phi_s_sa(geo,theta))
   {
      addRadialLeak(&flowVec,geo, Is1, Isa, phi_ie,phi_ie-theta,radialFlowModel, &
          count);
      addRadialLeak(&flowVec,geo, Is2, Isa, phi_ie,phi_ie-theta,radialFlowModel, &
          count);
      addRadialLeak(&flowVec,geo, Ic1[0], Isa, phi_ie-theta, phi_s_sa(geo,theta),
          radialFlowModel,&count);
      addRadialLeak(&flowVec,geo, Ic2[0], Isa, phi_ie-theta, phi_s_sa(geo,theta),
          radialFlowModel,&count);
      addRadialLeak(&flowVec,geo, Ic1[0], Is2, phi_s_sa(geo,theta),phi_ie-theta-PI
          ,radialFlowModel,&count);
      addRadialLeak(&flowVec,geo, Ic2[0], Is1, phi_s_sa(geo,theta),phi_ie-theta-PI
          ,radialFlowModel,&count);
   }
   else
   {
      addRadialLeak(&flowVec,geo, Is1, Isa, phi_ie,phi_s_sa(geo,theta),
          radialFlowModel, &count);
      addRadialLeak(&flowVec,geo, Is2, Isa, phi_ie,phi_s_sa(geo,theta),
          radialFlowModel, &count);
      addRadialLeak(&flowVec,geo, Is2, Is1, phi_s_sa(geo,theta),phi_ie-theta,
          radialFlowModel,&count);
      addRadialLeak(&flowVec,geo, Ic1[0], Is2, phi_ie-theta,phi_ie-theta-PI,
          radialFlowModel,&count);
      addRadialLeak(&flowVec,geo, Ic2[0], Is1, phi_ie-theta,phi_ie-theta-PI,
          radialFlowModel,&count);
   }
   //Compression Chamber leakages
   for (alpha=1;alpha<=Nc;alpha++)
   {
      addRadialLeak(&flowVec,geo, Ic1[alpha-1], Ic2[alpha-1], phi_ie-theta-2.0*PI
          *(alpha-1.0)-PI,phi_ie-theta-2.0*PI*(alpha),radialFlowModel,&count);
      //                                ^^^^ alpha-1 since C uses 0-based indexing
      if (alpha>1)
      {
         addRadialLeak(&flowVec,geo, Ic1[alpha-1], Ic2[alpha-2], phi_ie-theta-2.0*
             PI*(alpha-1.0),phi_ie-theta-2.0*PI*(alpha-1.0)-PI,radialFlowModel,&
             count);
         addRadialLeak(&flowVec,geo, Ic2[alpha-1], Ic1[alpha-2], phi_ie-theta-2.0*
             PI*(alpha-1.0),phi_ie-theta-2.0*PI*(alpha-1.0)-PI,radialFlowModel,&
             count);
      }
   }
   //Discharge chamber leakages
   // Caveat: does not take discharge region (across arcs or lines) into account
   if (useDDD)
   {
      addRadialLeak(&flowVec,geo, Iddd, Ic2[Nc-1], phi_ie-theta-2.0*PI*Nc,phi_ie-
          theta-2.0*PI*Nc-PI,radialFlowModel,&count);
      addRadialLeak(&flowVec,geo, Iddd, Ic1[Nc-1], phi_ie-theta-2.0*PI*Nc,phi_ie-
          theta-2.0*PI*Nc-PI,radialFlowModel,&count);
   }
   else
```

```
    {
        addRadialLeak(&flowVec,geo, Id1, Ic2[Nc-1], phi_ie-theta-2.0*PI*Nc,phi_ie-
            theta-2.0*PI*Nc-PI,radialFlowModel,&count);
        addRadialLeak(&flowVec,geo, Id2, Ic1[Nc-1], phi_ie-theta-2.0*PI*Nc,phi_ie-
            theta-2.0*PI*Nc-PI,radialFlowModel,&count);
        if (phi_ie-theta-2.0*PI*Nc-PI>geo->phi.phi_fis)
        {
        addRadialLeak(&flowVec,geo, Id2, Id1, phi_ie-theta-2.0*PI*Nc-PI,geo->phi.
            phi_fis,radialFlowModel,&count);
        }
    }
}
else
{
    addRadialLeak(&flowVec,geo, Is1, Isa, phi_ie, phi_s_sa(geo,theta),
        radialFlowModel,&count);
    addRadialLeak(&flowVec,geo, Is2, Isa, phi_ie, phi_s_sa(geo,theta),
        radialFlowModel,&count);
    addRadialLeak(&flowVec,geo, Is1, Is2, phi_s_sa(geo,theta), phi_ie-theta,
        radialFlowModel,&count);
    if (useDDD)
    {
        addRadialLeak(&flowVec,geo, Iddd, Is2, phi_ie-theta, phi_ie-theta-PI,
            radialFlowModel,&count);
        addRadialLeak(&flowVec,geo, Iddd, Is1, phi_ie-theta, phi_ie-theta-PI,
            radialFlowModel,&count);
    }
    else
    {
        addRadialLeak(&flowVec,geo, Id1, Is2, phi_ie-theta, phi_ie-theta-PI,
            radialFlowModel,&count);
        addRadialLeak(&flowVec,geo, Id2, Is1, phi_ie-theta, phi_ie-theta-PI,
            radialFlowModel,&count);
        addRadialLeak(&flowVec,geo, Id1, Id2, phi_ie-theta-PI, phi_ie-theta-2.0*PI,
            radialFlowModel,&count);
    }
}
//Fill flank values
if (Nc>0)
{
    addFlankLeak(&flowVec,geo,Is1,Ic1[0],flankFlowModel,&count);
    addFlankLeak(&flowVec,geo,Is2,Ic2[0],flankFlowModel,&count);
    if (Nc>1)
    {
        for (alpha=2;alpha<=Nc;alpha++)
        {
            addFlankLeak(&flowVec,geo,Ic1[alpha-1],Ic1[alpha-2],flankFlowModel,&count
                );
            addFlankLeak(&flowVec,geo,Ic2[alpha-1],Ic2[alpha-2],flankFlowModel,&count
                );
        }
    }
    if (useDDD)
    {
        addFlankLeak(&flowVec,geo,Iddd,Ic1[Nc-1],flankFlowModel,&count);
        addFlankLeak(&flowVec,geo,Iddd,Ic2[Nc-1],flankFlowModel,&count);
    }
    else
    {
        addFlankLeak(&flowVec,geo,Id1,Ic1[Nc-1],flankFlowModel,&count);
        addFlankLeak(&flowVec,geo,Id2,Ic2[Nc-1],flankFlowModel,&count);
    }
}
else
{
    if (useDDD)
    {
        addFlankLeak(&flowVec,geo,Is1,Iddd,flankFlowModel,&count);
        addFlankLeak(&flowVec,geo,Is2,Iddd,flankFlowModel,&count);
    }
    else
    {
        addFlankLeak(&flowVec,geo,Is1,Id1,flankFlowModel,&count);
        addFlankLeak(&flowVec,geo,Is2,Id2,flankFlowModel,&count);
    }
}
//Fill primary values
addPrimaryFlow(&flowVec,geo,Isa,Isuction,theta,"suction",suctionFlowModel,&count);
addPrimaryFlow(&flowVec,geo,Is1,Isa,theta,"s-sa",s_saFlowModel,&count);
addPrimaryFlow(&flowVec,geo,Is2,Isa,theta,"s-sa",s_saFlowModel,&count);
```

```
    if (useDDD==1)
    {
        addPrimaryFlow(&flowVec,geo,Iddd,Idischarge,theta,"discharge",
            dischargeFlowModel,&count);
    }
    else
    {
        addPrimaryFlow(&flowVec,geo,Idd,Id1,theta,"d-dd",d_ddFlowModel,&count);
        addPrimaryFlow(&flowVec,geo,Idd,Id2,theta,"d-dd",d_ddFlowModel,&count);
        addPrimaryFlow(&flowVec,geo,Idd,Idischarge,theta,"discharge",dischargeFlowModel
            ,&count);
    }
    /* Check whether you are at the discharge angle, and if so,
    change all the CV indices to the appropriate names */
    if (LeftDischarge==true)
    {
        for (i=0;i<count;i++)
        {
            if (flowVec.CV1[i]==Iddd)
                flowVec.CV1[i]=Idd;
            if (flowVec.CV2[i]==Iddd)
                flowVec.CV2[i]=Idd;
            if (flowVec.CVup[i]==Iddd)
                flowVec.CVup[i]=Idd;
            if (flowVec.CV1[i]==Ic1[nC_Max(geo)-1])
                flowVec.CV1[i]=Id1;
            if (flowVec.CV2[i]==Ic1[nC_Max(geo)-1])
                flowVec.CV2[i]=Id1;
            if (flowVec.CVup[i]==Ic1[nC_Max(geo)-1])
                flowVec.CVup[i]=Id1;
            if (flowVec.CV1[i]==Ic2[nC_Max(geo)-1])
                flowVec.CV1[i]=Id2;
            if (flowVec.CV2[i]==Ic2[nC_Max(geo)-1])
                flowVec.CV2[i]=Id2;
            if (flowVec.CVup[i]==Ic2[nC_Max(geo)-1])
                flowVec.CVup[i]=Id2;
        }
    }
    //Resize vectors
    flowVec.N=count;
    realloc(flowVec.CV1,     (count)*sizeof(int));
    realloc(flowVec.CV2,     (count)*sizeof(int));
    realloc(flowVec.CVup,       (count)*sizeof(int));
    realloc(flowVec.A,          (count)*sizeof(double));
    realloc(flowVec.flowModel, (count)*sizeof(int));
    realloc(flowVec.mdot,       (count)*sizeof(double));
    realloc(flowVec.mdot_L,     (count)*sizeof(double));
    realloc(flowVec.h_up,       (count)*sizeof(double));
    realloc(flowVec.h_down,     (count)*sizeof(double));
    realloc(flowVec.p_up,       (count)*sizeof(double));
    realloc(flowVec.p_down,     (count)*sizeof(double));
    realloc(flowVec.T_up,       (count)*sizeof(double));
    realloc(flowVec.T_down,     (count)*sizeof(double));
    realloc(flowVec.xL,         (count)*sizeof(double));
    realloc(flowVec.Ed,         (count)*sizeof(double));
    realloc(flowVec.Re,         (count)*sizeof(double));
    realloc(flowVec.Ma,         (count)*sizeof(double));
    return flowVec;
}
void freeFlowVec(struct flowVecVals *flowVec)
{
    free(flowVec->CV1);
    free(flowVec->CV2);
    free(flowVec->CVup);
    free(flowVec->A);
    free(flowVec->flowModel);
    free(flowVec->mdot);
    free(flowVec->mdot_L);
    free(flowVec->h_up);
    free(flowVec->h_down);
    free(flowVec->T_up);
    free(flowVec->T_down);
    free(flowVec->p_up);
    free(flowVec->p_down);
    free(flowVec->xL);
    free(flowVec->Ed);
    free(flowVec->Re);
    free(flowVec->Ma);
```

```
}
void setDiscGeo(struct geoVals *geo,char *Type)
{
   /*
   This function sets the discharge geometry parameters depending
   on the type of discharge geometry desired.  The following string
   values for the variable Type are defined:
   1) "ArcLineArc", which takes the radius of arc 1 and arc 2 from the
      disc input file
   2) "ArcLineArc-PMP", which takes the radius of arc 2 from the
      disc input file and determines the radius of arc 1 by first
      assuming that the phi_os=phi_is+pi and then solving for the
      PMP arc radius
   3) "2Arc", which takes the radius of arc 2 from the disc input file
      and then calculates the radius of arc 1
   4) "2Arc-PMP", which does not require any information from the disc
      input file
   */
   double a,b,c,x_is,y_is,x_os,y_os,nx_is,ny_is,nx_os,ny_os,dx,dy,r1,r2,r2_max,xarc1,
       xarc2,yarc1,yarc2,
      alpha,beta,d,L,xint,yint,xe,ye;

   // Common code for both ALA and 2Arc solutions
   coords_inv(geo->phi.phi_fis,0,geo,"fi",&x_is,&y_is);
   coords_inv(geo->phi.phi_fos,0,geo,"fo",&x_os,&y_os);
   coords_norm(geo->phi.phi_fis,"fi",&nx_is,&ny_is);
   coords_norm(geo->phi.phi_fos,"fo",&nx_os,&ny_os);
   dx=x_is-x_os;
   dy=y_is-y_os;
   // Maxiumum possible value of r2 for both perfect meshing and not perfect meshing
   a=cos(geo->phi.phi_fos-geo->phi.phi_fis)+1.0;
   b=geo->ro*a-dx*(sin(geo->phi.phi_fos)-sin(geo->phi.phi_fis))+dy*(cos(geo->phi.
       phi_fos)-cos(geo->phi.phi_fis));
   c=1.0/2.0*(2.0*dx*sin(geo->phi.phi_fis)*geo->ro-2.0*dy*cos(geo->phi.phi_fis)*geo->
       ro-dy*dy-dx*dx);
   if (geo->phi.phi_fos>geo->phi.phi_fis+PI)
      r2_max=(-b+sqrt(b*b-4.0*a*c))/(2.0*a);
   else //difference is equal to PI, it can't be less (collision otherwise)
      r2_max=-c/b;
   // First determine r2 if perfect meshing is being used for 2Arc solution
   if  (!strcmp(Type,"2Arc-PMP"))
      r2=r2_max;
   else // r2 value is passed in for either ArcLineArc w/ or w/o PMP, or 2Arc w/o PMP
      r2=geo->disc.ra_arc2;
   // If ALA, or ALA PMP, 2Arc PMP check that r2 is less than max possible
   if ( (!strcmp(Type,"ArcLineArc-PMP") || !strcmp(Type,"ArcLineArc") || !strcmp(Type
       ,"2Arc-PMP"))
      && (r2>r2_max+1e-12))
   {
      printf_plus("r2 value calculated [%g] is greater than max r2 possible [%g]\n",
          r2,r2_max);
      exit(EXIT_FAILURE);
   }
   // Then determine r1
   if (!strcmp(Type,"2Arc") || !strcmp(Type,"2Arc-PMP"))
   {
      // Covers both PMP and non-PMP cases
      r1=((1.0/2*dy*dy+1.0/2*dx*dx+r2*dx*sin(geo->phi.phi_fos)-r2*dy*cos(geo->phi.
          phi_fos))
        /(r2*cos(geo->phi.phi_fos-geo->phi.phi_fis)+dx*sin(geo->phi.phi_fis)-dy*cos(
            geo->phi.phi_fis)+r2));
   }
   else if (!strcmp(Type,"ArcLineArc-PMP"))
      r1=r2+geo->ro;
   else
      r1=geo->disc.ra_arc1;
   // Coordinates of the centers of the arcs
   xarc2 =  x_os+nx_os*r2;
   yarc2 =  y_os+ny_os*r2;
      // Inner starting normal has negative sign since
      // you want the outward pointing unit normal vector
   xarc1 =  x_is-nx_is*r1;
   yarc1 =  y_is-ny_is*r1;
   if (!strcmp(Type,"ArcLineArc") || !strcmp(Type,"ArcLineArc-PMP"))
   {
        geo->disc.xa_arc2=xarc2;
        geo->disc.ya_arc2=yarc2;
        geo->disc.ra_arc2=r2;
```

```
      geo->disc.t2_arc2=atan2(y_os-yarc2,x_os-xarc2);
      geo->disc.xa_arc1=xarc1;
      geo->disc.ya_arc1=yarc1;
      geo->disc.ra_arc1=r1;
      geo->disc.t2_arc1=atan2(y_is-yarc1,x_is-xarc1);
      alpha=atan2(yarc2-yarc1,xarc2-xarc1);
      d=sqrt(powInt(yarc2-yarc1,2)+powInt(xarc2-xarc1,2));
   beta=acos((r1+r2)/d);
      L=sqrt(d*d-(r1+r2)*(r1+r2));
      geo->disc.t1_arc1=alpha+beta;
   xint=xarc1+r1*cos(alpha+beta)+L*sin(alpha+beta);
   yint=yarc1+r1*sin(alpha+beta)-L*cos(alpha+beta);
      geo->disc.t1_arc2=atan2(yint-yarc2,xint-xarc2);
   /* Shift the angles to ensure that going from t1 to t2 proceeds
    in a counter-clockwise fashion, and takes less than 1 revolution */
      while (geo->disc.t2_arc2<geo->disc.t1_arc2)
          geo->disc.t2_arc2=geo->disc.t2_arc2+2.0*PI;
      while (geo->disc.t2_arc1<geo->disc.t1_arc1)
          geo->disc.t2_arc1=geo->disc.t2_arc1+2.0*PI;
      /*
      line given by y=m*t+b between the intersection points of the two circles
      */
      geo->disc.m_line=-1/tan(alpha+beta);
      geo->disc.t1_line=xarc1+r1*cos(geo->disc.t1_arc1);
      geo->disc.t2_line=xarc2+r2*cos(geo->disc.t1_arc2);
      geo->disc.b_line=yarc1+r1*sin(alpha+beta)-geo->disc.m_line*geo->disc.t1_line;
}
 if (!strcmp(Type,"2Arc") || !strcmp(Type,"2Arc-PMP"))
{
      geo->disc.xa_arc2=xarc2;
      geo->disc.ya_arc2=yarc2;
      geo->disc.ra_arc2=r2;
      geo->disc.t1_arc2=atan2(yarc1-yarc2,xarc1-xarc2);
      geo->disc.t2_arc2=atan2(y_os-yarc2,x_os-xarc2);
   /* Shift the angles to ensure that going from t1 to t2 proceeds
    in a counter-clockwise fashion, and takes less than 1 revolution */
      while (geo->disc.t2_arc2<geo->disc.t1_arc2)
          geo->disc.t2_arc2=geo->disc.t2_arc2+2.0*PI;
      geo->disc.xa_arc1=xarc1;
      geo->disc.ya_arc1=yarc1;
      geo->disc.ra_arc1=r1;
      geo->disc.t2_arc1=atan2(y_is-yarc1,x_is-xarc1);
      geo->disc.t1_arc1=atan2(yarc2-yarc1,xarc2-xarc1);
   /* Shift the angles to ensure that going from t1 to t2 proceeds
    in a counter-clockwise fashion, and takes less than 1 revolution */
      while (geo->disc.t2_arc1<geo->disc.t1_arc1)
          geo->disc.t2_arc1=geo->disc.t2_arc1+2.0*PI;
   /*
      line given by y=m*t+b with one element at the intersection
      point (with b=0, m=y/t)
      */
      geo->disc.b_line=0.0;
      geo->disc.t1_line=xarc2+r2*cos(geo->disc.t1_arc2);
      geo->disc.t2_line=geo->disc.t1_line;
      geo->disc.m_line=(yarc2+r2*sin(geo->disc.t1_arc2))/geo->disc.t1_line;
}
// If a negative value for disc_R is used in discharge input
// file, use that as a fraction of the largest possible port
// which is equal to the radius of arc 1
if (geo->disc.R<0.0)
{
   geo->disc.R=-geo->disc.R*geo->disc.ra_arc1;
   geo->disc.x0=geo->disc.xa_arc1;
   geo->disc.y0=geo->disc.ya_arc1;
   printf_plus("Discharge port radius is now: %g [m]\n",geo->disc.R);
}
// If a negative value for wall radius is passed in, determine
// the near minimal wall radius that can ensure no scroll-wall contact
// At theta=pi, the scrolls fill the largest possible circle whose radius is
// the distance from the outer ending angles of both scrolls with a center
// half the way to the origin for the orbiting scroll wrap
// The factor of 1.03 gives a bit of breathing room
if (geo->wall.r<0.0)
{
   geo->wall.x0=geo->ro/2.0*cos(geo->phi.phi_fie-PI/2-PI);
```

```
      geo->wall.y0=geo->ro/2.0*sin(geo->phi.phi_fie-PI/2-PI);
      coords_inv(geo->phi.phi_fie,PI,geo,"fo",&xe,&ye);
      geo->wall.r=1.03*sqrt(powInt(geo->wall.x0-xe,2)+powInt(geo->wall.y0-ye,2));
      printf_plus("Shell wall radius is now: %g [m]\n",geo->wall.r);
   }
}

// ************************************************************
//                    DISCHARGE PORT BLOCKAGE CODE
// ************************************************************
double A_disc(struct geoVals *geo,double theta)
{
   gpc_polygon disc_hole,diff,lobe;
   //FILE *fp;
   double x,y,*xdiff=NULL,*ydiff=NULL,*t=NULL,*phi_v=NULL,om,A=0;
   int i,k,I,Narc1,Nline,Narc2;
   om=geo->phi.phi_fie-PI/2.0-theta;

   //................................................
   //             Discharge port polygon
   //................................................
   disc_hole.num_contours=1;
   disc_hole.hole=NULL;
   disc_hole.contour=(gpc_vertex_list*)malloc(1*sizeof(gpc_vertex_list));
   disc_hole.contour->num_vertices=1000;
   disc_hole.contour->vertex=(gpc_vertex*)malloc(1000*sizeof(gpc_vertex));
   t=linspace(0,2*PI,1000);
   for (i=0;i<1000;i++)
   {
      disc_hole.contour->vertex[i].x=geo->disc.x0+geo->disc.R*cos(t[i]);
      disc_hole.contour->vertex[i].y=geo->disc.y0+geo->disc.R*sin(t[i]);
   }
   free(t);
   lobe.num_contours=1;
   lobe.hole=0;
   lobe.contour=(gpc_vertex_list*)malloc(1*sizeof(gpc_vertex_list));
   lobe.contour->num_vertices=500;
   lobe.contour->vertex=(gpc_vertex*)malloc(500*sizeof(gpc_vertex));
   // ..............................................
   // Depending on type of discharge, build the polygon
   // ..............................................
   if (!strcmp(geo->disc.Type,"Arc-Arc") || !strcmp(geo->disc.Type,"2Arc") || !strcmp
       (geo->disc.Type,"2Arc-PMP"))
   {  Narc1=150; Nline=0; Narc2=150;}
   if (!strcmp(geo->disc.Type,"Arc"))
   {  Narc1=300; Nline=0; Narc2=0;}
   if (!strcmp(geo->disc.Type,"Arc-Line-Arc") || !strcmp(geo->disc.Type,"ArcLineArc")
        || !strcmp(geo->disc.Type,"ArcLineArc-PMP"))
   {  Narc1=100; Nline=100; Narc2=100;}
   //................................................
   //Proceeding clockwise around the orbiting scroll
   //................................................
   // First is the outer involute segment with 250 elements
   phi_v=linspace(geo->phi.phi_oos+PI,geo->phi.phi_oos,100);
   for (i=0;i<100;i++)
   {
      coords_inv(phi_v[i],theta,geo,"oo",&x,&y);
      lobe.contour->vertex[i].x=x;
      lobe.contour->vertex[i].y=y;
   }
   free(phi_v);
   //Next the small arc
   if (Narc2>0)
   {
      t=linspace(geo->disc.t2_arc2,geo->disc.t1_arc2,Narc1);
      for (i=0;i<Narc1;i++)
      {
         lobe.contour->vertex[i+100].x=-geo->disc.xa_arc2-geo->disc.ra_arc2*cos(t[i])
             +geo->ro*cos(om);
         lobe.contour->vertex[i+100].y=-geo->disc.ya_arc2-geo->disc.ra_arc2*sin(t[i])
             +geo->ro*sin(om);
      }
      free(t);
   }
   //Next the line segment
   if (Nline>0)
   {
```

```
        I=100+Narc2;
        t=linspace(geo->disc.t2_line,geo->disc.t1_line,Nline);
        for (i=0;i<Nline;i++)
        {
            lobe.contour->vertex[i+I].x=-t[i]+geo->ro*cos(om);
            lobe.contour->vertex[i+I].y=-geo->disc.m_line*t[i]-geo->disc.b_line+geo->ro*
                sin(om);
        }
        free(t);
    }
    //Next the big arc
    if (Narc1>0)
    {
        I=100+Narc2+Nline;
        t=linspace(geo->disc.t1_arc1,geo->disc.t2_arc1,Narc1);
        for (i=0;i<Narc1;i++)
        {
            lobe.contour->vertex[i+I].x=-geo->disc.xa_arc1-geo->disc.ra_arc1*cos(t[i])+
                geo->ro*cos(om);
            lobe.contour->vertex[i+I].y=-geo->disc.ya_arc1-geo->disc.ra_arc1*sin(t[i])+
                geo->ro*sin(om);
        }
        free(t);
    }
    // Last is the outer involute segment with 100 elements
    I=100+Narc2+Nline+Narc1;
    phi_v=linspace(geo->phi.phi_ois,geo->phi.phi_ois+PI/20.0,100);
    for (i=0;i<100;i++)
    {
        coords_inv(phi_v[i],theta,geo,"oi",&x,&y);
        lobe.contour->vertex[i+400].x=x;
        lobe.contour->vertex[i+400].y=y;
    }
    free(phi_v);
    //fp=fopen("disc_hole.csv", "wb");
    //gpc_write_polygon(fp,0,&disc_hole);
    //fclose(fp);
    //fp=fopen("lobe.csv", "wb");
    //gpc_write_polygon(fp,0,&lobe);
    //fclose(fp);
    gpc_polygon_clip(GPC_DIFF,&disc_hole,&lobe,&diff);
    for (k=0;k<diff.num_contours;k++)
    {
        xdiff=calloc(diff.contour[k].num_vertices,sizeof(double));
        ydiff=calloc(diff.contour[k].num_vertices,sizeof(double));
        for (i=0;i<diff.contour[k].num_vertices;i++)
        {
            xdiff[i]=diff.contour[k].vertex[i].x;
            ydiff[i]=diff.contour[k].vertex[i].y;
        }
        A+=polyArea(xdiff,ydiff,diff.contour[k].num_vertices-1);
        free(xdiff);
        free(ydiff);
    }
    //fp=fopen("disc_hole.csv", "wb");
    //gpc_write_polygon(fp,0,&disc_hole);
    //fclose(fp);
    //fp=fopen("lobe.csv", "wb");
    //gpc_write_polygon(fp,0,&lobe);
    //fclose(fp);
    //fp=fopen("diff.csv", "wb");
    //gpc_write_polygon(fp,0,&diff);
    //fclose(fp);
    // Free dynamically allocated polygon data
    gpc_free_polygon(&disc_hole);
    gpc_free_polygon(&lobe);
    gpc_free_polygon(&diff);
    return A;
}
double interpVec(double *t, double *A, double t_goal, int N)
{
//
    double t0,t1,t2,L0,L1,L2,f0,f1,f2;
    int L,M,R;
    L=0;
```

```
    R=N-1;
    M=(L+R)/2;
    // Use interval halving to find the indices which bracket the angle of interest
    while (R-L>1)
    {
       if (t_goal>=t[M])
       {
          L=M;
          M=(L+R)/2;
       }
       if (t_goal<t[M])
       {
          R=M;
          M=(L+R)/2;
       }
    }
    if (L<N-2)
    {
       t0=t[L];
       t1=t[L+1];
       t2=t[L+2];
       f0=A[L];
       f1=A[L+1];
       f2=A[L+2];
    }
    else
    {
       t0=t[R-2];
       t1=t[R-1];
       t2=t[R];
       f0=A[R-2];
       f1=A[R-1];
       f2=A[R];
    }
    L0=(t_goal-t1)*(t_goal-t2)/((t0-t1)*(t0-t2));
    L1=(t_goal-t0)*(t_goal-t2)/((t1-t0)*(t1-t2));
    L2=(t_goal-t0)*(t_goal-t1)/((t2-t0)*(t2-t1));
    return L0*f0+L1*f1+L2*f2;
}

//*************************************************************
//                    Geometry Driver Function
//*************************************************************
void GeometryModel(struct geoVals *geo,double theta, int Istep, int useDDD, int
    LeftDischarge,
                double *V, double *dV, struct flowVecVals * flowVec)
{
    int i;
    double Vd, dVd, Vdd, dVdd,cx,cy;
    flowVec[Istep]=buildFlowVec(geo,theta,useDDD,LeftDischarge);
    V[Isa+NCV*Istep]=Vsa_integrate(geo,theta);
    dV[Isa+NCV*Istep]=dVsa_integrate(geo,theta);
    Vs1_calcs(geo,theta,&(V[Is1+NCV*Istep]),&(dV[Is1+NCV*Istep]),&cx,&cy);
    V[Is2+NCV*Istep]=V[Is1+NCV*Istep];
    dV[Is2+NCV*Istep]=dV[Is1+NCV*Istep];
    for (i=0;i<nC(geo,theta);i++)
    {
       Vc1_calcs(geo,theta,i+1,&(V[Ic1[i]+NCV*Istep]),&(dV[Ic1[i]+NCV*Istep]),&cx,&cy)
           ;//i+1 alpha since alpha starts at 1
       V[Ic2[i]+NCV*Istep]=V[Ic1[i]+NCV*Istep];
       dV[Ic2[i]+NCV*Istep]=dV[Ic1[i]+NCV*Istep];
    }
    //new method
    Vd1_calcs(geo,theta,&Vd,&dVd,&cx,&cy);
    Vdd_calcs(geo,theta,&Vdd,&dVdd,&cx,&cy);
    if (useDDD==1)
    {
       V[Iddd+NCV*Istep]=2*Vd+Vdd;
       dV[Iddd+NCV*Istep]=2*dVd+dVdd;
    }
    else
    {
       V[Id1+NCV*Istep]=Vd;
       V[Id2+NCV*Istep]=Vd;
       V[Idd+NCV*Istep]=Vdd;
       dV[Id1+NCV*Istep]=dVd;
       dV[Id2+NCV*Istep]=dVd;
```

```
      dV[Idd+NCV*Istep]=dVdd;
   }
   if (LeftDischarge==true)
   {
      if (theta<theta_d(geo))
      {
         // Use the compression chamber volume (innermost one)
         Vc1_calcs(geo,theta,nC(geo,theta),&Vd,&dVd,&cx,&cy);
         V[Id1+NCV*Istep]=Vd;
         V[Id2+NCV*Istep]=Vd;
         dV[Id1+NCV*Istep]=dVd;
         dV[Id2+NCV*Istep]=dVd;
      }
      else
      {
         //Use the dd chamber volume
         V[Id1+NCV*Istep]=Vd;
         V[Id2+NCV*Istep]=Vd;
         dV[Id1+NCV*Istep]=dVd;
         dV[Id2+NCV*Istep]=dVd;
      }
      V[Ic1[nC_Max(geo)-1]+NCV*Istep]=0.0;
      V[Ic2[nC_Max(geo)-1]+NCV*Istep]=0.0;
      dV[Ic1[nC_Max(geo)-1]+NCV*Istep]=0.0;
      dV[Ic2[nC_Max(geo)-1]+NCV*Istep]=0.0;
      V[Iddd+NCV*Istep]=0.0;
      dV[Iddd+NCV*Istep]=0.0;
   }
}
void cleanUpGeo()
{
   free(Ic1);
   Ic1=NULL;
   free(Ic2);
   Ic2=NULL;
   free(Iinjection);
   Iinjection=NULL;
}
```

MassFlow.h

```
// MassFlow.h
#ifndef MASS_FLOW
#define MASS_FLOW
void Two_Phase_Nozzle(char * Gas, char* Liq, double A, double w_ent, double sigma,
    double P_1, double P_2, double T_1, double xL, double *T_2, double *mdot, double
    *Re, double *Ma);
void Bends(char * Gas, char* Liq, double A, double z_D, double w_sa_s, double
    h_scroll, double T_1, double P_1, double xL, double P_2, double *T_2, double *
    mdot, double *Re, double *Ma);
void Tee(char * Gas, char* Liq, double A, double T_1, double P_1, double xL, double
    P_2, double *T_2, double *mdot, double *Re, double *Ma);
void Dry_Gas_Friction(char * Gas, char* Liq, double A, double L, double delta, double
     T_1, double P_1, double P_2, double *T_2, double *mdot, double *Re, double *Ma);
void Frictional_Flank_Leakage(char * Ref, char *Liq, double A, double R, double r,
    double phi_flank, double delta_r, double T_up, double p_up, double p_down,double
    *mdot, double *Re, double *Ma);
void CalcMassFlows(struct scrollVals *scroll, int Itheta);
void TestFlowModels();
#endif
```

MassFlow.c

```
#ifndef __GNUC__
#define _CRTDBG_MAP_ALLOC
#include <stdlib.h>
#include <crtdbg.h>
#else
#include <stdlib.h>
#endif
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "StructsMacros.h"
#include "Geo/geoFuncs.h"
```

```
#include "Props/FloodProp.h"
#include "Props/R744.h"
#include "MyFuncs.h"
#include "ScrollsModel.h"
#include "MassFlow.h"
#include "time.h"
void Two_Phase_Nozzle(char * Gas, char* Liq, double A, double w_ent, double sigma,
    double P_1, double P_2, double T_1, double xL, double *T_2, double *mdot, double
    *Re, double *Ma)
{
   double I,dP,v_l,v_g,v_g0,K_e,v_e_1,v_e_high,P,xG,e_t,G_max,G_thr,M,beta,v_e_thr,T,
       gamma,C_c_1,C_d0;
   double f,kN2,C_dg,Prat,C_dL,w,v_h_t,v_e_2,dI;
   int kk,N=20;
   I=0;
   gamma=c_p(Gas,T_1,P_1)/c_v(Gas,T_1,P_1);
   T=T_1;
    dP=(P_1-P_2)/N;
    P=P_1;
    v_l=1/rho_l(Liq,T);
    v_g=1/rho_g(Gas,T,P);
    v_g0=v_g;
    xG=1-xL;
    K_e=cK_e(v_l,v_g,xG,w_ent,1.0);
    v_e_1=cv_e(v_l,v_g,K_e,xG,w_ent,1.0);
    v_e_high=v_e_1;
    for(kk=1;kk<=N;kk++)
    {
        P=P-dP;// [kPa]
        v_g=1/rho_g(Gas,T,P); //[m^3/kg]
        K_e=cK_e(v_l,v_g,xG,w_ent,1.0); //[-]
        v_e_2=cv_e(v_l,v_g,K_e,xG,w_ent,1.0); //[m^3/kg]
        dI=dP/2.0*(v_e_1+v_e_2);
        I=I+dI;
        v_e_1=v_e_2;
    }
    /* Two-Phase Discharge Coefficient from Morris */
    beta=sqrt(sigma);
    v_e_thr=v_e_2;
    Prat=P_2/P_1;
    v_h_t=xG*v_g0*pow(Prat,-1/gamma)+(1-xG)*v_l;
    C_dL=0.6135+0.13318*pow(beta,2)-0.26095*pow(beta,4)+0.51146*pow(beta,6);
    f=1/C_dL-1/(2*pow(C_dL,2));
    kN2=2*gamma/(gamma-1)*pow(Prat,2/gamma)*(1-pow(Prat,(gamma-1)/gamma));
    w=4*pow(Prat,2/gamma)*(1-Prat)*f/kN2;
    C_dg=(1-pow(1-w,0.5))/(2*f*pow(Prat,1/gamma));
    e_t=1/(1+(1-xG)/xG*v_l/v_g0*pow(Prat,1/gamma)*pow(v_h_t/v_l,0.5));
    C_d0=e_t*C_dg+(1-e_t)*C_dL;
    C_c_1=(1.26-0.26*beta)*C_d0;
    G_thr=sqrt(2.0*I/(pow(v_e_thr,2.0)-pow(beta,4.0)*pow(v_e_1,2.0))*1000.0);
   /*Two-Phase Choking*/
    G_max=sqrt(-1000.0/(xG*dvdP_m(Gas,Liq,T,P_2,0)+(1-xG)*dvdP_m(Gas,Liq,T,P_2,1)));
    M=G_thr*sqrt((-xG*dvdP_m(Gas,Liq,T,P_2,0)-(1-xG)*dvdP_m(Gas,Liq,T,P_2,1))/1000.0)
        ;
    if (M>1)
    {
        G_thr=G_max;
    }
   *T_2=T_1;
   *mdot=G_thr*A;
   *Ma=M;
   *Re=G_thr*sqrt(4.0*A/PI)/mu_mix(Gas,Liq,T_1,P_1,1.0-xG);
   if (*mdot>100000)
   {
      printf_plus("Mass flow rate in Two_Phase_Nozzle too high\n");
   }
}
void Tee(char * Gas, char* Liq, double A, double T_1, double P_1, double xL, double
    P_2, double *T_2, double *mdot, double *Re, double *Ma)
{
   double eps, change,muf,vf,vg,Dh,Re_Lo,f,DPb;
   double B_180,phi2,DP2,G_thr,x=0.,x1=0.,x2=0.,x3=0.,y1=0.,y2=0.,G=0.;
   int iter;
   // Tee at inlet
    eps=1e-3;
    change=999;
    iter=1;
    while (iter<=3 || change>eps){
```

```
        if (iter==1){ x1=1000; G=x1; }
        if (iter==2){ x2=1100; G=x2; }
        if(iter>2)
            G=x2;
        muf=mu_mix(Gas,Liq,T_1,(P_1+P_2)/2,1);
        vf=1/rho_m(Gas,Liq,T_1,(P_1+P_2)/2,1);
        vg=1/rho_m(Gas,Liq,T_1,(P_1+P_2)/2,0);
        x=1-xL;
        Dh=sqrt(A*4.0/PI);
        Re_Lo=G*Dh/muf;
        if (Re_Lo<1038)
        {
             f=16.0/Re_Lo;
        }
         else
        {
            f=0.35/4.0*pow(Re_Lo,-0.25);
        }
        DPb=pow(G,2.0)*vf/2*1.25; // Chisholm Eq 1.10 & 10.4
        B_180=1.8; // Chisholm p. 160 Table
        phi2=1.0+(vg/vf-1.0)*(B_180*x*(1.0-x)+pow(x,2.0));
        //1.6 factor from upstream disturbance
        DP2=phi2*DPb*1.6-(P_1-P_2)*1000;
        if (iter==1){y1=DP2;}
        if (iter==2){y2=DP2;}
        if (iter>2){
            y2=DP2;
            x3=x2-y2/(y2-y1)*(x2-x1);
            change=fabs(y2/(y2-y1)*(x2-x1));
            y1=y2; x1=x2; x2=x3;
        }
        iter=iter+1;
        if (iter>40)
            printf_plus("x1: %g \t x2: %g \t y1: %g \t y2: %g \t \n",x1,x2,y1,y2);
        if (iter>100)
         printf_plus("tee mdot not converging");
    }
    G_thr=x3;
    *T_2=T_1;
   *mdot=G_thr*A;
   *Ma=G_thr*sqrt((-x*dvdP_m(Gas,Liq,T_1,P_2,0)-(1-x)*dvdP_m(Gas,Liq,T_1,P_2,1))
       /1000.0);
   *Re=G*sqrt(4.0*A/PI)/mu_mix(Gas,Liq,T_1,P_1,1-x);
}
void Bends(char * Gas, char* Liq, double A, double z_D, double w_sa_s, double
    h_scroll, double T_1, double P_1, double xL, double P_2, double *T_2, double *
    mdot, double *Re, double *Ma)
{
   double eps, change,muf,vf,vg,Dh,Re_Lo,f,theta,DPb,DPg;
   double k,B_90,B_180,phi2,DP2,G_thr,x=0.,x1=0.,x2=0.,x3=0.,y1=0.,y2=0.,G=0.;
   int iter;
   // Bends for suction
   eps=1e-3;
   change=999;
   iter=1;
   while (iter<=3 || change>eps){
      if (iter==1){x1=1000; G=x1;}
      if (iter==2){x2=1100; G=x2;}
      if(iter>2){G=x2;}
      muf=mu_mix(Gas,Liq,T_1,(P_1+P_2)/2,1);
      vf=1/rho_m(Gas,Liq,T_1,(P_1+P_2)/2,1);
      vg=1/rho_m(Gas,Liq,T_1,(P_1+P_2)/2,0);
      x=1-xL;
      Dh=2.0*w_sa_s*h_scroll/(w_sa_s+h_scroll);
      Re_Lo=G*Dh/muf;
      f=0.314/4.0/pow(Re_Lo,0.25);
      theta=180.0;
      DPb=2.0*f*pow(1.0-x,2.0)*pow(G,2.0)*vf*z_D*(2.0*theta/180.0);
      DPg=2.0*f*pow(x,2.0)*pow(G,2.0)*vg*z_D*(2.0*theta/180.0);
      k=4.0*f*z_D;
      B_90=1.0+2.2/(k*(2.0+0.5));
      B_180=0.5*(1.0+B_90);
      phi2=1/pow(1.0-x,2.0)*(1.0+(vg/vf-1.0)*(B_180*x*(1.0-x)+pow(x,2.0)));
      if (x==0.0)
         DP2=phi2*DPb*1.6-(P_1-P_2)*1000;
      else
         DP2=DPg-(P_1-P_2)*1000;
```

```
        if (iter==1){y1=DP2;}
        if (iter==2){y2=DP2;}
        if (iter>2)
        {
            y2=DP2;
            x3=x2-y2/(y2-y1)*(x2-x1);
            change=fabs(y2/(y2-y1)*(x2-x1));
            y1=y2;
            x1=x2;
            x2=x3;
        }
        iter=iter+1;
        if (iter>40){
            printf_plus("x1: %g \t x2: %g \t y1: %g \t y2: %g \t \n",x1,x2,y1,y2);
        }
        if (iter>100){
        printf_plus("bends not converging");
        }
    }
    G_thr=x3;
    *T_2=T_1;
    *mdot=G_thr*A;
    *Ma=G_thr*sqrt((-x*dvdP_m(Gas,Liq,T_1,P_2,0)-(1-x)*dvdP_m(Gas,Liq,T_1,P_2,1))
        /1000.0);
    *Re=G*sqrt(4.0*A/PI)/mu_mix(Gas,Liq,T_1,P_1,1-x);
}
void Dry_Gas_Friction(char * Gas, char* Liq, double A, double L, double delta, double
     T_1, double P_1, double P_2, double *T_2, double *mdot, double *Re_out, double *
     Ma_out)
{
    // Leaks with friction
    double Dh,G_thr,mu,v,a,b,G_laminar,Re_laminar;
    /*
    First assume the flow is laminar, and f is therefore:
    f=16/Re, so if f=a*Re^b, a = 16, b = -1 (Fanning friction factor)
    */
    a=16.0; b=-1.0;
    //a=0.35/4.0; b=-0.25; //From Ishii, 1996
    mu=mu_g(Gas,T_1,(P_1+P_2)/2.0);
    v=1/rho_g(Gas,T_1,(P_1+P_2)/2.0);
    Dh=2*delta;
    G_laminar=pow(pow(mu,b)*(P_1-P_2)*1000.0/(2*a*pow(Dh,b-1.0)*v*L),1.0/(b+2.0));
    Re_laminar=G_laminar*Dh/mu;
    G_thr=G_laminar;
    *mdot=G_thr*A;
    *Ma_out=G_thr*sqrt(-dvdP_m(Gas,Liq,T_1,P_2,0.0)/1000.0);
    *Re_out=Re_laminar;
}
void Liquid_Leakage_Friction(char * Gas, char* Liq, double A, double L, double delta,
     double T_1, double P_1, double P_2, double *T_2, double *mdot, double *Re_out,
     double *Ma_out)
{
    // Leaks with friction
    double Dh,G_thr,mu,v,a,b,G_laminar,Re_laminar;
    /*
    First assume the flow is laminar, and f is therefore:
    f=16/Re, so if f=a*Re^b, a = 16, b = -1 (Fanning friction factor)
    */
    a=16.0; b=-1.0;
    mu=mu_l(Liq,T_1);
    v=1/rho_l(Liq,T_1);
    Dh=2*delta;
    G_laminar=pow(pow(mu,b)*(P_1-P_2)*1000.0/(2*a*pow(Dh,b-1.0)*v*L),1.0/(b+2.0));
    Re_laminar=G_laminar*Dh/mu;
    G_thr=G_laminar;
    *mdot=G_thr*A;
    *Ma_out=G_thr*sqrt(-dvdP_m(Gas,Liq,T_1,P_2,1.0)/1000.0);
    *Re_out=Re_laminar;
}
void Frictional_Flank_Leakage(char * Ref, char *Liq, double A, double R_, double r,
     double phi_flank, double delta_r, double T_up, double p_up, double p_down,double
     *mdot, double *Re, double *Ma)
{
    int N_int=30,i;
    double phi1,phi2,h1,h2,Integral,Integrand1,Integrand2,dphi;
    double alpha, beta,rho,nu,DELTAp,u_m,gamma;
    DELTAp=(p_up-p_down)*1000.0;  // 1000 for conversion from kPa to Pa
    Integral=0;
```

```
    dphi=phi_flank/(N_int-1);
    rho=rho_m(Ref,Liq,T_up,p_up,0.0);
    nu=mu_mix(Ref,Liq,T_up,p_up,0.0)/rho;
    gamma=c_p(Ref,T_up,p_up)/c_v(Ref,T_up,p_up);
    for (i=0;i<N_int;i++)
    {
        phi1=i*dphi;
        h1=R_-(R_-r-delta_r)*cos(phi1)-sqrt(r*r-(R_-r-delta_r)*(R_-r-delta_r)*sin(phi1)
            *sin(phi1));
        Integrand1=1.0/(h1*h1*h1);    // 1/h^3
        phi2=(i+1)*dphi;
        h2=R_-(R_-r-delta_r)*cos(phi2)-sqrt(r*r-(R_-r-delta_r)*(R_-r-delta_r)*sin(phi2)
            *sin(phi2));
        Integrand2=1.0/(h2*h2*h2);
        Integral+=(Integrand1+Integrand2)/2.0*dphi;
    }
    Integral*=2; // The integral from -phi_flank to phi_flank equals twice the
        integral from 0 to phi_flank by symmetry
    //Blasius smooth pipe
    /*alpha=0.315;
    beta=-0.25;*/
    alpha=3.74;
    beta=-0.45;
    u_m=pow(pow(2.0,beta)*alpha*R_*rho*Integral/(4.0*DELTAp*pow(nu,beta)),-1.0/(2.0+
        beta))/delta_r;
    *Re=u_m*delta_r*2.0/nu;
    *mdot=u_m*rho*A;
    *Ma=u_m/sqrt(gamma*R(Ref)*1000*T_up);
}
void IsentropicDryIdealGas(char * Gas,double A, double delta, double t, double ro,
    double T_up, double p_up, double p_down, int Type, double *T_down, double *mdot,
    double *Re, double *Ma)
{
    double k,R,rho_up,v,Dh,mu,c,rho_down,Lstar,delta_star;
    double a_radial[]={25932.1070099 , 0.914825434095 , -177.588568125 ,
        -0.237052788124 , -172347.610527 , -12.0687599808 , -0.0128861161041 ,
        -151.202604262 , -0.999674457769 , 0.0161435039267 , 0.825533456725};
    double Re_star_radial=5243.58194594;
    double a_flank[]={-2.63970395918 , -0.567164431229 , 0.83655499929 ,
        0.810567167521 , 6174.02825667 , -7.60907962464 , -0.510200923053 ,
        -1205.17482697 , -1.02938914174 , 0.689497785772 , 1.09607735134};
    double Re_star_flank=826.167177885;
    double *a,Re_star,xi,mdot_ratio;
    // Since ideal, R=cp-cv, and k=cp/cv
        R=(c_p(Gas,T_up,p_up)-c_v(Gas,T_up,p_up))*1000.0;
        k=c_p(Gas,T_up,p_up)/c_v(Gas,T_up,p_up);
    // Speed of sound
        c=sqrt(k*R*T_up);
    //Upstream density
        rho_up=p_up*1000.0/(R*T_up);
    if ( p_down/p_up > pow(1.+(k-1)/2.,k/(1-k)) )
    {
        // Mass flow rate if not choked
            *mdot=A*p_up*1000.0/(sqrt(R*T_up))*sqrt(2*k/(k-1.0)*pow(p_down/p_up,2.0/k)
                *(1-pow(p_down/p_up,(k-1.0)/k)));
        // Throat temperature
            *T_down=T_up*pow(p_down/p_up,(k-1.)/k);
        // Throat density
            rho_down=p_down*1000.0/(R* *T_down);
        // Velocity at throat
            v= *mdot /(rho_down*A);
        // Mach number
            *Ma =v/c;
    }
    else
    {
        // Mass flow rate if choked
            *mdot=A*rho_up*sqrt(k*R*T_up)*pow(1.+(k-1.)/2.,(1+k)/(2*(1-k)));
        // Velocity at throat
            v= c;
        //Mach Number
            *Ma=1.0;
    }
    // Hydraulic diameter
        Dh=2.*delta;
    // Viscosity for Re
```

```
        mu=mu_g(Gas,T_up,p_up);
    // Reynolds number
        *Re=rho_up*v*Dh/mu;
    if (Type==CORRECTED_RADIAL_NOZZLE)
    {
        a=a_radial;
        Re_star=Re_star_radial;
        xi=1.0/(1.0+exp(-0.01*(*Re-Re_star)));
        Lstar=t/0.005;
        delta_star=delta/10e-6;
        mdot_ratio=a[0]*pow(Lstar,a[1])/(a[2]*delta_star+a[3])*(xi*(a[4]*pow(*Re,a[5])+
            a[6])+(1-xi)*(a[7]*pow(*Re,a[8])+a[9]))+a[10];
    }
    else if (Type==CORRECTED_FLANK_NOZZLE)
    {
        a=a_flank;
        Re_star=Re_star_flank;
        xi=1.0/(1.0+exp(-0.01*(*Re-Re_star)));
        Lstar=ro/0.005;
        delta_star=delta/10e-6;
        mdot_ratio=a[0]*pow(Lstar,a[1])/(a[2]*delta_star+a[3])*(xi*(a[4]*pow(*Re,a[5])+
            a[6])+(1-xi)*(a[7]*pow(*Re,a[8])+a[9]))+a[10];
    }
    else
    {
        mdot_ratio=1.0;
    }
    *mdot=*mdot/mdot_ratio;
}
void CalcMassFlows(struct scrollVals *scroll, int Itheta)
{
    int i,I1,I2,flowModel,smallSuction,k,Nc;
    double R=0.,r=0.,A,T_up,p_up,p_down,xL,Z_D,delta,L,w_ent,sigma,L_flank,phi_flank;
    double *T_down, *mdot,flowSign,*Re,*Ma, theta ,t,ro;
    //printf_plus("In mdot");
    for (i=0;i<scroll->flowVec[Itheta].N;i++)
    {
        /* Define matrix indices */
        I1=scroll->flowVec[Itheta].CV1[i]+NCV*Itheta;
        I2=scroll->flowVec[Itheta].CV2[i]+NCV*Itheta;
        /* Find the the upstream control volume */
        if (  (scroll->p[I1]) > (scroll->p[I2])  )
        {
            /* Then the pressure of CV1 is greater than
            that in CV2 and the flow is from CV1 to CV2 */
            p_up=scroll->p[I1];
            p_down=scroll->p[I2];
            T_up=scroll->T[I1];
            xL=scroll->xL[I1];
            scroll->flowVec[Itheta].CVup[i]=scroll->flowVec[Itheta].CV1[i];
            /* Flow sign is defined based on CV1, so since flow
            is leaving CV1, the mass flow will be given as negative
            */
            flowSign=-1.0;
        }
        else
        {
            /* Then the pressure of CV1 is less than
            that in CV2 and the flow is from CV2 to CV1 */
            p_up=scroll->p[I2];
            p_down=scroll->p[I1];
            T_up=scroll->T[I2];
            xL=scroll->xL[I2];
            scroll->flowVec[Itheta].CVup[i]=scroll->flowVec[Itheta].CV2[i];
            /* Flow sign is defined based on CV1, so since flow
            is entering CV1, the mass flow will be given as positive
            */
            flowSign=+1.0;
        }
        /* Make alias pointers to memory locations */
        A=scroll->flowVec[Itheta].A[i];
        flowModel=scroll->flowVec[Itheta].flowModel[i];
        mdot=&(scroll->flowVec[Itheta].mdot[i]);
        T_down=&(scroll->flowVec[Itheta].T_down[i]);
        Re=&(scroll->flowVec[Itheta].Re[i]);
        Ma=&(scroll->flowVec[Itheta].Ma[i]);
        if (flowModel==DRY_GAS_FLANK_FLANK_MODEL)
        {
            /* This block calculates the radii of curvature of the outer and innner
```

```
    involutes which form a flow path */
    if (scroll->flowVec[Itheta].CVup[i]==Is1 || scroll->flowVec[Itheta].CVup[i
        ]==Is2)
    {
        // Flow from s1 or s2 to sa
        theta=scroll->theta[Itheta];
        R=scroll->geo.rb*(scroll->geo.phi.phi_fie-theta-scroll->geo.phi.phi_fi0);
        r=scroll->geo.rb*(scroll->geo.phi.phi_foe-PI-theta-scroll->geo.phi.
            phi_fo0);
    }
    else
    {
        if (scroll->flowVec[Itheta].CVup[i]==Iddd || scroll->flowVec[Itheta].CVup
            [i]==Idd
            || scroll->flowVec[Itheta].CVup[i]==Id1 || scroll->flowVec[Itheta].
                CVup[i]==Id2)
        {
            // Flow from ddd or dd to respective compression chamber
            theta=scroll->theta[Itheta];
            Nc=nC(&(scroll->geo),theta);
            R=scroll->geo.rb*(scroll->geo.phi.phi_fie-theta-2.0*PI*Nc-scroll->geo.
                phi.phi_fi0);
            r=scroll->geo.rb*(scroll->geo.phi.phi_foe-PI-theta-2.0*PI*Nc-scroll->
                geo.phi.phi_fo0);
        }
        else
        {
            // From from compression chamber to suction chamber or
            // from compression chamber to compression chamber
            theta=scroll->theta[Itheta];
            for (k=0;k<nC(&(scroll->geo),theta);k++)
            {
                if (scroll->flowVec[Itheta].CVup[i]==Ic1[k] || scroll->flowVec[
                    Itheta].CVup[i]==Ic2[k])
                {
                    R=scroll->geo.rb*(scroll->geo.phi.phi_fie-theta-2.0*PI*(k+1)-
                        scroll->geo.phi.phi_fi0);
                    r=scroll->geo.rb*(scroll->geo.phi.phi_foe-PI-theta-2.0*PI*(k+1)-
                        scroll->geo.phi.phi_fo0);
                }
            }
        }
    }
}
smallSuction=false;
/* If either the upstream or downstream CV are suction chambers and
the flow is leakage flow, and you are still at the beginning of the rotation,
turn off mass flow so that you avoid driving the oil mass fraction below zero
    */
if ((flowModel==DRY_GAS_RADIAL_FRICTIONAL_MODEL || flowModel==
    DRY_GAS_FLANK_FRICTIONAL_MODEL) &&
    (scroll->flowVec[Itheta].CV1[i]==Is1 || scroll->flowVec[Itheta].CV1[i]==Is2
        ||
    scroll->flowVec[Itheta].CV2[i]==Is1 || scroll->flowVec[Itheta].CV2[i]==Is2)
        &&
    scroll->theta[Itheta]<2.0*PI/10.0 && (scroll->xL[I1]<0.01 || scroll->xL[I2
        ]<0.01)
    && scroll->xL[Is1]>0.0 && scroll->xL[Is1]>0.0)
    smallSuction=true;
//printf_plus("-past alias -");
scroll->flowVec[Itheta].T_up[i]=T_up;
scroll->flowVec[Itheta].p_up[i]=p_up;
scroll->flowVec[Itheta].p_down[i]=p_down;
/*Define terms */
Z_D=scroll->massFlow.Inputs.Z_D_bends;
w_ent=scroll->massFlow.Inputs.w_ent;
sigma=scroll->massFlow.Inputs.sigma;
L_flank=scroll->massFlow.Inputs.L_flank;
phi_flank=scroll->massFlow.Inputs.phi_flank; //Limit of +- phi_flank for
    integration for flank leakage
if (p_up-p_down>0.0001 && A>0 && smallSuction==false)
{
    switch (flowModel)
    {
        case TWO_PHASE_NOZZLE:
            scroll->flowVec[Itheta].xL[i]=xL; // Use CV oil fraction
            Two_Phase_Nozzle(scroll->Ref,scroll->Liq,A,w_ent,sigma,p_up,p_down,
                T_up,xL,T_down,mdot,Re,Ma);
            break;
        case BENDS_MODEL:
```

```
                scroll->flowVec[Itheta].xL[i]=xL; // Use CV oil fraction
                Bends(scroll->Ref,scroll->Liq,A,Z_D,A/(scroll->geo.hs),scroll->geo.hs,
                    T_up,p_up,xL,p_down,T_down,mdot,Re,Ma);
                break;
            case CORRECTED_RADIAL_NOZZLE:
                scroll->flowVec[Itheta].xL[i]=0.0; // Use dry gas
                delta=scroll->massFlow.Inputs.delta_radial;
                t=scroll->geo.rb*(scroll->geo.phi.phi_fi0-scroll->geo.phi.phi_fo0);
                ro=scroll->geo.rb*PI-t;
                IsentropicDryIdealGas(scroll->Ref,A,delta,t,ro,T_up,p_up,p_down,
                    flowModel,T_down,mdot,Re,Ma);
                break;
            case CORRECTED_FLANK_NOZZLE:
                scroll->flowVec[Itheta].xL[i]=0.0; // Use dry gas
                delta=scroll->massFlow.Inputs.delta_flank;
                t=scroll->geo.rb*(scroll->geo.phi.phi_fi0-scroll->geo.phi.phi_fo0);
                ro=scroll->geo.rb*PI-t;
                IsentropicDryIdealGas(scroll->Ref,A,delta,t,ro,T_up,p_up,p_down,
                    flowModel,T_down,mdot,Re,Ma);
                break;
            case TEE_FLOW_MODEL:
                scroll->flowVec[Itheta].xL[i]=xL; // Use CV oil fraction
                Tee(scroll->Ref,scroll->Liq,A,T_up,p_up,xL,p_down,T_down,mdot,Re,Ma);
                break;
            case DRY_GAS_RADIAL_FRICTIONAL_MODEL:
                scroll->flowVec[Itheta].xL[i]=0.0; // Use dry gas
                delta=scroll->massFlow.Inputs.delta_radial;
                L=scroll->geo.t;
                Dry_Gas_Friction(scroll->Ref,scroll->Liq,A,L,delta,T_up,p_up,p_down,
                    T_down,mdot,Re,Ma);
                break;
            case DRY_GAS_FLANK_FRICTIONAL_MODEL:
                scroll->flowVec[Itheta].xL[i]=0.0; // Use dry gas
                delta=scroll->massFlow.Inputs.delta_flank;
                L=L_flank;
                Dry_Gas_Friction(scroll->Ref,scroll->Liq,A,L,delta,T_up,p_up,p_down,
                    T_down,mdot,Re,Ma);
                break;
            case LIQUID_RADIAL_FRICTIONAL_MODEL:
                scroll->flowVec[Itheta].xL[i]=1.0; // Use all oil
                delta=scroll->massFlow.Inputs.delta_radial;
                L=scroll->geo.t;
                Liquid_Leakage_Friction(scroll->Ref,scroll->Liq,A,L,delta,T_up,p_up,
                    p_down,T_down,mdot,Re,Ma);
                break;
            case LIQUID_FLANK_FRICTIONAL_MODEL:
                scroll->flowVec[Itheta].xL[i]=1.0; // Use all oil
                delta=scroll->massFlow.Inputs.delta_flank;
                L=L_flank;
                Liquid_Leakage_Friction(scroll->Ref,scroll->Liq,A,L,delta,T_up,p_up,
                    p_down,T_down,mdot,Re,Ma);
                break;
            case DRY_GAS_FLANK_FLANK_MODEL:
                scroll->flowVec[Itheta].xL[i]=0.0; // Use dry gas
                delta=scroll->massFlow.Inputs.delta_flank;
                Frictional_Flank_Leakage(scroll->Ref,scroll->Liq,A,R,r,phi_flank,delta
                    ,T_up,p_up,p_down,mdot,Re,Ma);
                break;
        }
        *mdot *= flowSign;
        *T_down=T_up;
        if (isNAN(*mdot) || isINFINITY(*mdot) || fabs(*mdot)>100000)
            printf_plus("Mdot not real number");
    }
    else
    {
        *T_down=T_up;
    }
    scroll->flowVec[Itheta].h_up[i]=h_m(scroll->Ref,scroll->Liq,T_up,p_up,xL);
    scroll->flowVec[Itheta].h_down[i]=h_m(scroll->Ref,scroll->Liq,*T_down,p_down,xL
        );
  }
}
```

HeatTransfer.h

```
/* File HeatTransfer.h */
 #ifndef HEATTRANSFER_H
 #define HEATTRANSFER_H
```

```
void TubeHT(char * Ref, char *Liq, double T_in, double p_in, double xL_in, double L,
         double D, double mdot, double T_wall, /*in  -- out */ double *Q, double *
             T_out);
int HTphi(struct scrollVals *scroll, int ICV, double theta, /*in ---- out */
            double *phi_1_i,double *phi_2_i, double *phi_1_o, double *phi_2_o);
void scrollHT(struct scrollVals *scroll, double theta, int Itheta, double T_scroll);
double Qwall(double hc, double hs, double rb, double phi1, double phi2, double phi0,
          double T_scroll, double T_CV, double dT_dphi, double phim);
 #endif
```

HeatTransfer.c

```
#ifndef __GNUC__
   #define _CRTDBG_MAP_ALLOC
   #include <stdlib.h>
   #include <crtdbg.h>
#else
   #include <stdlib.h>
#endif
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "Geo/geoFuncs.h"
#include "MyFuncs.h"
#include "Props/FloodProp.h"
#include "ScrollsModel.h"
#include "time.h"
#include "HeatTransfer.h"
#include "StructsMacros.h"
// Private function prototypes
double findFlow(struct scrollVals *scroll, int Itheta,int ICV1, int ICV2);
void TubeHT(char * Ref, char *Liq, double T_in, double p_in, double xL_in, double L,
         double D, double mdot, double T_wall, /*in  -- out */ double *Q, double *
             T_out)
{
   // Flow through a tube with the wall at a constant temperature
   // Assumes flow is turbulent and fully developed
   double Pr,Re,hc,k,cp;
    Pr=Pr_mix(Ref,Liq,T_in,p_in,xL_in); //[-]
   Re=4*mdot/(PI*mu_mix(Ref,Liq,T_in,p_in,xL_in)*D); //[-]
   k=k_mix(Ref,Liq,T_in,p_in,xL_in); //[kW/m-K]
   cp=cp_mix(Ref,Liq,T_in,p_in,xL_in); //[kJ/kg-K]
    hc=0.023*k/D*pow(Re,0.8)*pow(Pr,0.4); //[kW/m^2-K]
   *T_out=T_wall-(T_wall-T_in)*exp(-PI*D*L*hc/(mdot*cp));
   // Q is defined to be positive if heat transferred from wall to fluid
   *Q=mdot*cp*(*T_out-T_in);
}
double findFlow(struct scrollVals *scroll, int Itheta,int ICV1, int ICV2)
{
   // Search through a flowVec vector at a given step i
   // and look for a CV1/CV2 pairing and return its mass flow rate
   int j;
   for (j=0;j < scroll->flowVec[Itheta].N;j++)
   {
      if (scroll->flowVec[Itheta].CV1[j]==ICV1 &&
         scroll->flowVec[Itheta].CV2[j]==ICV2)
      {
         return scroll->flowVec[Itheta].mdot[j];
      }
   }
   // Can't find the CV pair, and return a huge number
   return _HUGE;
}
void scrollHT(struct scrollVals *scroll, double theta, int Itheta, double T_scroll)
{
   double dT_dphi,phim,phi_i0,phi_o0;
   char *Ref, *Liq;
   double T,p,xL,V;
   double Pr,Re,hc,Dh,mdot;
   double A_plate,A_wall_in, A_wall_out, phi_1_i, phi_2_i, phi_1_o, phi_2_o;
   double T_1_i, T_2_i, T_1_o, T_2_o, T_plate_m;
   double T_avg, p_avg,xL_avg,f,Amax,St,rho,r_c,Ubar;
   double Q_wall_i,Q_wall_o, Q_plate;
   int useCV,ICV;
   // ------------------------------------------------------
   // For the suction chambers, the limits are
```

```
// ending angle and the first conjugate angle.
// For the compression chambers, the limits
// are the initial and final conjugate angles.
// For the discharge chamber (d1 or d2), the final
// angle is the starting angle, and the initial angle is the
// innermost conjugate point
// Scroll heat transfer is neglected for the discharge region,
// but plate heat transfer is included.
// ------------------------------------------------
// dT_dphi is negative because as you move to the
// outside of the scroll (larger phi), the temperature goes down because
// you are moving towards the suction temperature
dT_dphi=( scroll->T[Isuction] - scroll->T[Idischarge] ) /
   ( scroll->geo.phi.phi_fie - scroll->geo.phi.phi_fos );
phim=( scroll->geo.phi.phi_fie + scroll->geo.phi.phi_fos )/2.0;
// From symmetry, phi_i0=phi_fi0=phi_oi0 , and same for o0
phi_i0=scroll->geo.phi.phi_fi0;
phi_o0=scroll->geo.phi.phi_fo0;
Ref=&(scroll->Ref);
Liq=&(scroll->Liq);
for (ICV=0;ICV<NCV;ICV++)
{
   //Retrieve values from matrix
   T=scroll->T[ICV+NCV*Itheta];
   p=scroll->p[ICV+NCV*Itheta];
   xL=scroll->xL[ICV+NCV*Itheta];
   V=scroll->V[ICV+NCV*Itheta];
   // phi_1 is larger than phi_2
   useCV=HTphi(scroll,ICV,theta,&phi_1_i,&phi_2_i,&phi_1_o,&phi_2_o);
   if (useCV==true)
   {
      // Assume that the mass flow is the total mass flow
      mdot=scroll->massFlow.mdot_tot;
      A_plate = V/( scroll->geo.hs );
      A_wall_in = scroll->geo.hs * scroll->geo.rb * ( (powInt(phi_1_i,2)-powInt(
          phi_2_i,2) )/2.0 -
            phi_i0*(phi_1_i-phi_2_i) );
      A_wall_out = scroll->geo.hs * scroll->geo.rb * ( (powInt(phi_1_o,2) -powInt(
          phi_2_o,2) )/2.0 -
            phi_o0*(phi_1_o-phi_2_o) );
      if (scroll->flags.useDDD==true && ICV==Iddd)
      {
         // Double the wall areas since the two d1 and d2 parts contribute
         // Does not take into account the wall area from the arcs and lines
         // in the discharge area
         A_wall_in*=2; A_wall_out*=2;
      }
      Dh=4.0*scroll->geo.ro*scroll->geo.hs/(2.0*scroll->geo.ro+scroll->geo.hs);
      T_avg=(scroll->T[Isuction]+scroll->T[Idischarge])/2.0;
      p_avg=(scroll->p[Isuction]+scroll->p[Idischarge])/2.0;
      xL_avg=(scroll->xL[Isuction]);
      rho=rho_m(Ref,Liq,T_avg,p_avg,xL_avg); //[-]
      Pr=Pr_mix(Ref,Liq,T_avg,p_avg,xL_avg); //[-]
      Re=4.0*mdot/2.0/(PI*mu_mix(Ref,Liq,T_avg,p_avg,xL_avg)*Dh); //[-]
      hc=0.023*k_mix(Ref,Liq,T_avg,p_avg,xL_avg)/Dh*pow(Re,0.8)*pow(Pr,0.4); //[kW
          /m^2-K]
      // Jang and Jeong correction for spiral geometry
      f=scroll->omega/(2*PI);
      Amax=scroll->geo.ro;
      Ubar=scroll->massFlow.mdot_tot/(4*scroll->geo.ro*scroll->geo.hs*rho);
      St=f*Amax/Ubar;
      hc*=1.0+8.48*(1-exp(-5.35*St));
      // Tagri and Jayaraman correction for transverse oscillation
      r_c=scroll->geo.rb*(0.5*phi_1_i+0.5*phi_2_i-scroll->geo.phi.phi_fi0);
      hc*=1.0+1.77*Dh/r_c;
      //hc=0.0; //[kW/m^2-K]
      // Since the mean plate temperature for each CV is conduction controlled
      // based on the scroll temperatures as well as convective heat transfer with
      // the refrigerant, it is fair to assume that it can be calculated by
          averaging
      // the temperatures of both scrolls at the angles which define the
      // outer and inner involute portions of the scroll pocket
      T_1_i=T_scroll+dT_dphi*(phi_1_i-phim);
```

```
            T_2_i=T_scroll+dT_dphi*(phi_2_i-phim);
            T_1_o=T_scroll+dT_dphi*(phi_1_o-phim);
            T_2_o=T_scroll+dT_dphi*(phi_2_o-phim);
            T_plate_m = (T_1_i + T_2_i + T_1_o + T_2_o)/4.0;
            // Calculate the wall and plate heat transfer amounts
            Q_wall_i=Qwall(hc,scroll->geo.hs,scroll->geo.rb,phi_1_i,phi_2_i,phi_i0,
                T_scroll,T,dT_dphi,phim);
            Q_wall_o=Qwall(hc,scroll->geo.hs,scroll->geo.rb,phi_1_o,phi_2_o,phi_o0,
                T_scroll,T,dT_dphi,phim);
            Q_plate=2*hc*A_plate*(T_plate_m-T);
            // Sum up the heat transfers and store back in the scroll structure
            scroll->Q[ICV+NCV*Itheta]=Q_plate+Q_wall_i+Q_wall_o;
            // Store some other useful ancillary values
            scroll->HT.hc[ICV+NCV*Itheta]=hc;
            scroll->HT.A_wall_i[ICV+NCV*Itheta]=A_wall_in;
            scroll->HT.A_wall_o[ICV+NCV*Itheta]=A_wall_out;
            scroll->HT.Tm_plate[ICV+NCV*Itheta]=T_plate_m;
            scroll->HT.Tm_wall_i[ICV+NCV*Itheta]=T_scroll+dT_dphi*((phi_1_i+phi_2_i)
                /2.0-phim);
            scroll->HT.Tm_wall_o[ICV+NCV*Itheta]=T_scroll+dT_dphi*((phi_1_o+phi_2_o)
                /2.0-phim);
        }
        else
        {
            scroll->Q[ICV+NCV*Itheta]=0.0;
        }
    }
}
int HTphi(struct scrollVals *scroll, int ICV, double theta, /*in ---- out */
           double *phi_1_i, double *phi_2_i, double *phi_1_o, double *phi_2_o)
{
    int Nc,i;
    Nc=nC(&(scroll->geo),theta);
    if (ICV==Is1)
    {
        *phi_1_i=scroll->geo.phi.phi_fie;
        *phi_2_i=scroll->geo.phi.phi_fie-theta;
        *phi_1_o=phi_s_sa(&(scroll->geo),theta);
        *phi_2_o=scroll->geo.phi.phi_ooe-PI-theta;
        return true;
    }
    if (ICV==Is2)
    {
        *phi_1_i=scroll->geo.phi.phi_oie;
        *phi_2_i=scroll->geo.phi.phi_oie-theta;
        *phi_1_o=phi_s_sa(&(scroll->geo),theta);
        *phi_2_o=scroll->geo.phi.phi_foe-PI-theta;
        return true;
    }
    if (ICV==Id1 && scroll->flags.useDDD==false)
    {
        *phi_1_i=scroll->geo.phi.phi_fie-theta-Nc*2*PI;
        *phi_2_i=scroll->geo.phi.phi_fis;
        *phi_1_o=scroll->geo.phi.phi_ooe-PI-theta-Nc*2*PI;
        *phi_2_o=scroll->geo.phi.phi_oos;
        if(scroll->flags.LeftDischarge)
        {
            // At the Left of Discharge angle, you are still
            // before the discharge angle, so there is still one
            // more discharge chamber, but you want to consider the
            // compression chamber as part of the discharge chamber
            *phi_1_i += 2*PI;
            *phi_1_o += 2*PI;
        }
        return true;
    }
    if (ICV==Id2 && scroll->flags.useDDD==false)
    {
        *phi_1_i=scroll->geo.phi.phi_oie-theta-Nc*2*PI;
        *phi_2_i=scroll->geo.phi.phi_ois;
        *phi_1_o=scroll->geo.phi.phi_foe-PI-theta-Nc*2*PI;
        *phi_2_o=scroll->geo.phi.phi_fos;
        if(scroll->flags.LeftDischarge)
        {
            // At the Left of Discharge angle, you are still
            // before the discharge angle, so there is still one
            // more discharge chamber, but you want to consider the
            // compression chamber as part of the discharge chamber
```

```
            *phi_1_i += 2*PI;
            *phi_1_o += 2*PI;
        }
        return true;
    }
    if (ICV==Idd && scroll->flags.useDDD==false)
    {
        *phi_1_i=scroll->geo.phi.phi_ois;
        *phi_2_i=scroll->geo.phi.phi_ois;
        *phi_1_o=scroll->geo.phi.phi_fos;
        *phi_2_o=scroll->geo.phi.phi_fos;
        return true;
    }
    if (ICV==Iddd && scroll->flags.useDDD==true)
    {
        //Angles are for one half of the scroll area
        *phi_1_i=scroll->geo.phi.phi_oie-theta-Nc*2*PI;
        *phi_2_i=scroll->geo.phi.phi_ois;
        *phi_1_o=scroll->geo.phi.phi_foe-PI-theta-Nc*2*PI;
        *phi_2_o=scroll->geo.phi.phi_fos;
        return true;
    }
    if (Nc>0)
    {
        for (i=0;i<Nc;i++)
        {
            if (ICV==Ic1[i] && scroll->flags.LeftDischarge==false)
            {
                *phi_1_i=scroll->geo.phi.phi_fie-theta-i*2*PI;
                *phi_2_i=scroll->geo.phi.phi_fie-theta-(i+1)*2*PI;
                *phi_1_o=scroll->geo.phi.phi_ooe-PI-theta-i*2*PI;
                *phi_2_o=scroll->geo.phi.phi_ooe-PI-theta-(i+1)*2*PI;
                return true;
            }
            if (ICV==Ic2[i] && scroll->flags.LeftDischarge==false)
            {
                *phi_1_i=scroll->geo.phi.phi_oie-theta-i*2*PI;
                *phi_2_i=scroll->geo.phi.phi_oie-theta-(i+1)*2*PI;
                *phi_1_o=scroll->geo.phi.phi_foe-PI-theta-i*2*PI;
                *phi_2_o=scroll->geo.phi.phi_foe-PI-theta-(i+1)*2*PI;
                return true;
            }
        }
    }
    /*CV not found, so return false*/
    return false;
}
double Qwall(double hc, double hs, double rb, double phi1, double phi2, double phi0,
          double T_scroll, double T_CV, double dT_dphi, double phim)
{
    /* This function evaluates the anti-derivative of
    the differential of wall heat transfer, and returns the amount of scroll-
    wall heat transfer
    phi1 and phi2 are defined such that phi1 is always the
    larger involute angle
    */
    double term2,term1;
    term1=hc*hs*rb*( (phi1*phi1/2.0-phi0*phi1)*(T_scroll-T_CV)
        +dT_dphi*(phi1*phi1*phi1/3.0-(phi0+phim)*phi1*phi1/2.0+phi0*phim*phi1));
    term2=hc*hs*rb*( (phi2*phi2/2.0-phi0*phi2)*(T_scroll-T_CV)
        +dT_dphi*(phi2*phi2*phi2/3.0-(phi0+phim)*phi2*phi2/2.0+phi0*phim*phi2));
    return term1-term2;
}
```

ScrollsModel.h

```
// ScrollsModel.h
#ifndef SCROLLS_MODEL
#define SCROLLS_MODEL
    static int Ntheta;
    double calcP(struct scrollVals * scroll, int Itheta);
    struct scrollVals Initialize_ScrollModel(struct geoVals *geo, struct
        scrollInputVals *Inputs, struct ExperVals *Exper, struct MLVals *ML, struct
        FlowVals *Flows);
    void Initialize_Masses(struct scrollVals *scroll);
    void Initialize_Rotation(struct scrollVals *scroll, struct scrollInputVals *Inputs
        );
    void freeScroll(struct scrollVals *scroll);
    double *Derivs(struct scrollVals * scroll, int Itheta, double theta);
```

```
   double *dT_dp_dxL(struct scrollVals * scroll, int Itheta, double theta);
   double *dT_dm_dxL(struct scrollVals * scroll, int Itheta, double theta);
   void AtDischarge(struct scrollVals *scroll, int Itheta,double theta);
   void Merge(struct scrollVals *scroll, int Itheta,double theta);
   void Wrap_Rotation(struct scrollVals *scroll, struct scrollInputVals *Inputs);
   double Wrap_Error(struct scrollVals *scroll);
   void Rotation_RK45(struct scrollVals *scroll, struct scrollInputVals *Inputs);
   double newTd(struct scrollVals *scrollPtr, double *mdot_out, double *mdot_in);
   void CalculateLossTerms(struct scrollVals *scroll);
   void CalculateForces(struct scrollVals *scroll, int N);
#endif
```

ScrollsModel.c

```
//scrollsModel.c
#ifndef __GNUC__
#define _CRT_SECURE_NO_WARNINGS
#define _CRTDBG_MAP_ALLOC
#include <stdlib.h>
#include <crtdbg.h>
#endif
#include <stdio.h>
#include <string.h>
#include "math.h"
#include "time.h"
#include "StructsMacros.h"
#include "MyFuncs.h"
#include "Geo/geoFuncs.h"
#include "ScrollsModel.h"
#include "Props/FloodProp.h"
#include "MassFlow.h"
#include "HeatTransfer.h"
#include "SaveOutputs.h"

//Private function prototypes:
double sumQ(struct scrollVals *scroll);

int dtCounter;
int dTmmCounter;

// Change this and recompile to change working variables
int ODEVars = T_m_xL;
//int ODEVars = T_p_xL;

double OBJECTIVE_Td(double Td, struct scrollVals *scroll, struct scrollInputVals *
    Inputs)
{
   double mdot_suct, mdot_disc, Td_new,Td_old;
   Td_old=Td;
   Inputs->T_out=Td;
   Rotation_RK45(scroll,Inputs);
   Td_new=newTd(scroll, &(scroll->massFlow.mdot_disc),&(scroll->massFlow.mdot_suct));
   printf_plus("\tTd_old: %0.5f error: %0.5f\n",Inputs->T_out,Td_new-Inputs->T_out);
   printf_plus("\tmdot_disc: %0.5f, mdot_suct: %0.5f error[%%]: %0.5f\n",
      fabs(scroll->massFlow.mdot_disc),fabs(scroll->massFlow.mdot_suct),(fabs(scroll
          ->massFlow.mdot_disc)-fabs(scroll->massFlow.mdot_suct))/fabs(scroll->
          massFlow.mdot_suct)*100.0);
   scroll->Debug.Td_error_abs=Td_new-Td_old;
   Inputs->T_out=Td_new;
   return Td_new-Td_old;
}
double Secant_Td(double Td_guess, double delta, double eps,struct scrollVals *scroll,
    struct scrollInputVals *Inputs)
{
   double x1,x2,x3,y1,y2,change,f,T;
   int iter;
   change=999;
   iter=1;
   while ((iter<=3 || change>eps) && iter<100)
   {
      if (iter==1){x1=Td_guess; T=x1;}
      if (iter==2){x2=Td_guess+delta; T=x2;}
      if (iter>2) {T=x2;}
         f=OBJECTIVE_Td(T,scroll,Inputs);
      if (iter==1){y1=f;}
      if (iter>1)
      {
         y2=f;
         x3=x2-y2/(y2-y1)*(x2-x1);
         change=fabs(y2/(y2-y1)*(x2-x1));
         y1=y2; x1=x2; x2=x3;
```

```
      }
      iter=iter+1;
   }
   return T;
}
double Dekker_Td(double Td_min, double Td_max, double eps,struct scrollVals *scroll,
    struct scrollInputVals *Inputs)
{
   double a_k,b_k,f_ak,f_bk,f_bkn1,error,
      b_kn1,b_kp1,s,m,a_kp1,f_akp1,f_bkp1,x;
   long sign_bk,sign_bkn1;
   int iter=1;
   // Loop for the solver method
    while ((iter <= 1 || fabs(error) > eps) && iter < 100)
   {
      // Start with the maximum value
        if (iter == 1)
      {
            a_k = Td_max;
            x = a_k;
      }
      // End with the minimum value
        if (iter == 2)
      {
            b_k = Td_min;
            x = b_k;
      }
        if (iter > 2)
            x = b_k;
      // Evaluate residual
        error = OBJECTIVE_Td(x,scroll,Inputs);
      // First time through, store the outputs
        if(iter == 1)
      {
            f_ak = error;
            b_kn1 = a_k;
            f_bkn1 = error;
         if (fabs(error) < eps)
            b_k=a_k;
      }
        if (iter > 1)
      {
            f_bk = error;
         // Check if point and contrapoint have the same sign, if so, increase band
             and try again
         if (iter==2 && f_bk*f_bkn1>0)
         {
            if (f_bk<0)
            {
               printf_plus("In Dekker solver for Td, point and contrapoint give \n\t
                   same sign of residual, decreasing \n\t the temperature band by 5K\
                   n");
               Td_max=Td_min;
               Td_min=Td_min-5.0;
            }
            else
            {
               printf_plus("In Dekker solver for Td, point and contrapoint give \n\t
                   same sign of residual, increasing \n\t the temperature band by 5K\
                   n");
               Td_min=Td_max;
               Td_max=Td_max+5.0;
            }
            iter=1;
            continue;
         }
         //Secant solution
            s = b_k - (b_k - b_kn1) / (f_bk - f_bkn1) * f_bk;
         //Midpoint solution
            m = (a_k + b_k) / 2.0;
            if (s > b_k && s < m)
         {
              //Use the secant solution
              b_kp1 = s;
         }
            else
         {
            //Use the midpoint solution
                b_kp1 = m;
         }
            //See if the signs of iterate and contrapoint are the same
```

```
            f_bkp1 = OBJECTIVE_Td(b_kp1,scroll,Inputs);
         if (fabs(f_bkp1)<eps)
         {
            return b_kp1;
         }
            if (f_ak*f_bkp1<0)
         {
                // If a and b have opposite signs,
            //  keep the same contrapoint
                a_kp1 = a_k;
                f_akp1 = f_ak;
         }
            else
         {
                //Otherwise, keep the iterate
                a_kp1 = b_k;
                f_akp1 = f_bk;
         }
            if (fabs(f_akp1) < fabs(f_bkp1))
         {
                //a_k+1 is a better guess than b_k+1, so swap a and b values
            swap(&a_kp1, &b_kp1);
                swap(&f_akp1, &f_bkp1);
            printf_plus("Swapping point and contrapoint\n");
         }
            //Update variables
            //Old values
            b_kn1 = b_k;
            f_bkn1 = f_bk;
            //values at this iterate
            b_k = b_kp1;
            a_k = a_kp1;
            f_ak = f_akp1;
            f_bk = f_bkp1;
      }
        iter++;
      if (iter>90 && fabs(error)>eps)
      {
         printf("Dekker for Td has failed\n");
      }
   }
   OBJECTIVE_Td(b_k,scroll,Inputs);
   return b_k;
}
double Brent_Td(double Td_min, double Td_max, double eps,struct scrollVals *scroll,
    struct scrollInputVals *Inputs)
{
   double a,b,c,d,s,fa,fb,fc,fd,fs,delta;
   int mFlag=0;
   a=Td_min;
   fa=OBJECTIVE_Td(a,scroll,Inputs);
   b=Td_max;
   fb=OBJECTIVE_Td(b,scroll,Inputs);
   if (fa*fb>0)
   {
      printf_plus("%s\n","Doesn't bracket");
   }
   if (fabs(fa)<fabs(fb))
   {
      swap(&a,&b);
      swap(&fa,&fb);
   }
   c=a;
   fc=fa;
   mFlag=1;
   delta=eps;
   // Loop for the solver method
    while (fabs(b-a) > eps)
   {
      if (fa!=fc && fb!=fc)
      {
         printf_plus("\tQuadratic interpolation\n");
         printf_plus("\t%12.4f %12.4f %12.4f\n",a,b,c);
         printf_plus("\t%12.4f %12.4f %12.4f\n",fa,fb,fc);
         s=a*fb*fc/((fa-fb)*(fa-fc))+b*fa*fc/((fb-fa)*(fb-fc))+c*fa*fb/((fc-fa)*(fc-
             fb));
      }
      else
      {
         printf_plus("\tSecant interpolation\n");
```

```
            printf_plus("\t%12.4f %12.4f %12.4f\n",a,b,c);
            printf_plus("\t%12.4f %12.4f %12.4f\n",fa,fb,fc);
            s=b-fb*(b-a)/(fb-fa);
        }
        if (!(s>(3*a + b)/4 && s<b)
            || (mFlag==1 && fabs(s-b)>fabs(b-c)/2)
            || (mFlag==0 && fabs(s-b)>fabs(c-d)/2)
            || (mFlag==1 && fabs(b-c)<delta)
            || (mFlag==0 && fabs(c-d)<delta)    )
        {
            printf_plus("\tBisection interpolation\n");
            s=(a+b)/2;
            mFlag=1;
        }
        else
            mFlag=0;
        fs=OBJECTIVE_Td(s,scroll,Inputs);
        d=c;
        fd=fc;
        c=b;
        fc=fb;
        if (fa*fs<0.0)
        {
            b=s;
            fb=fs;
        }
        else
        {
            a=s;
            fa=fs;
        }
        if (fabs(fa)<fabs(fb))
        {
            swap(&fa,&fb);
            swap(&a,&b);
        }
    }
    return b;
}
double OBJECTIVE_Tscroll(double T_scroll,double Td, struct scrollVals *scroll, struct
     scrollInputVals *Inputs, struct scrollInputVals *UpstreamInputs)
{
    double TT,Td_resid;
    double L_tube,D_tube,Tout,kstar,Td_ideal,Td_max,Td_min,err_HT,h_disc,h_suct,m,b,c,
        mu_suct,mu_0,P_gas,halfBandWidth;
    printf_plus("T_scroll: %0.5f\n",T_scroll);
    scroll->HT.T_scroll=T_scroll;
    // Suction tube heating at inlet to compressor
    L_tube=scroll->massFlow.Inputs.L_inlet;
    D_tube=scroll->massFlow.Inputs.D_inlet;
    TubeHT(UpstreamInputs->Ref,UpstreamInputs->Liq,
        UpstreamInputs->T_in,UpstreamInputs->p_in,UpstreamInputs->xL_in,
        L_tube,D_tube,scroll->massFlow.mdot_tot,scroll->HT.T_scroll,
        &(scroll->HT.Q_scroll_inlet),&Tout);
    //Update the inlet temperature to the scroll model
    Inputs->T_in=Tout;
    Inputs->T_out=Td;
    //halfBandWidth=scroll->solver.Td_halfBandWidth;
    //printf_plus("Hopefully Td is in the range [%g,%g]\n",Inputs->T_out-halfBandWidth
        ,Inputs->T_out+halfBandWidth);
    OBJECTIVE_Td(Td,scroll,Inputs);
    //Dekker_Td(Inputs->T_out-halfBandWidth,Inputs->T_out+halfBandWidth,1e-3,scroll,
        Inputs);
    //Brent_Td(Inputs->T_out-halfBandWidth,Inputs->T_out+halfBandWidth,1e-2,scroll,
        Inputs);
    //Secant_Td(Inputs->T_out,0.05,1e-3,scroll,Inputs);
    scroll->HT.Q_scroll_gas = sumQ(scroll);
    scroll->HT.Q_scroll_amb = -scroll->ML.UA_amb*(scroll->HT.T_scroll-scroll->HT.T_amb
        );
    h_disc=h_m(scroll->Ref,scroll->Liq,Inputs->T_out,scroll->p[Idischarge],scroll->xL[
        Idischarge]);
    h_suct=h_m(scroll->Ref,scroll->Liq,scroll->T[Isuction],scroll->p[Isuction],scroll
        ->xL[Isuction]);
    mu_suct=mu_mix(scroll->Ref,scroll->Liq,scroll->T[Isuction],scroll->p[Isuction],
        scroll->xL[Isuction]);
    mu_0=mu_mix(scroll->Ref,scroll->Liq,300.0,300,0.0);
    P_gas=scroll->massFlow.mdot_suct*(h_disc-h_suct)-scroll->HT.Q_scroll_gas-scroll->
        HT.Q_scroll_inlet;
    if (!strcmp(scroll->ML.Type,"eta_m"))
```

```
    {
       m=(1-scroll->ML.eta_m)/scroll->ML.eta_m; b=0.0; c=0.0;
       scroll->ML.m=m;
    }
    else
    {
       m=scroll->ML.m;
       b=scroll->ML.b;
       c=scroll->ML.c;
    }
    scroll->PowerEff.P_ML=b+m*P_gas*pow(mu_suct/mu_0,c);
    scroll->PowerEff.P_gas=P_gas;
    L_tube=scroll->massFlow.Inputs.L_inlet;
    D_tube=scroll->massFlow.Inputs.D_inlet;
    TubeHT(Inputs->Ref,Inputs->Liq,
       Inputs->T_out,Inputs->p_out,Inputs->xL_in,
       L_tube,D_tube,scroll->massFlow.mdot_tot,scroll->HT.T_scroll,
       &(scroll->HT.Q_scroll_outlet),&Tout);
    err_HT=-scroll->HT.Q_scroll_inlet-scroll->HT.Q_scroll_outlet-scroll->HT.
        Q_scroll_gas+scroll->HT.Q_scroll_amb+scroll->PowerEff.P_ML;
    printf_plus("HT Error [kW]: %0.4f T_scroll: %0.5f \n",err_HT,scroll->HT.T_scroll);
    scroll->massFlow.mdot_tot=scroll->massFlow.mdot_suct;
    scroll->Debug.LumpHT_error_abs=err_HT;
    return err_HT;
}
double Dekker_Tscroll(double Ts_min, double Ts_max, double Td, double eps,struct
    scrollVals *scroll, struct scrollInputVals *Inputs, struct scrollInputVals *
    UpstreamInputs)
{
    double a_k,b_k,f_ak,f_bk,f_bkn1,error,
       b_kn1,b_kp1,s,m,a_kp1,f_akp1,f_bkp1,x;
    long sign_bk,sign_bkn1;
    int iter=1;
    // Loop for the solver method
     while ((iter <= 1 || fabs(error) > eps) && iter < 100)
    {
       // Start with the minimum value
         if (iter == 1)
       {
             a_k = Ts_min;
             x = a_k;
       }
       // End with the maximum value
         if (iter == 2)
       {
             b_k = Ts_max;
             x = b_k;
       }
         if (iter > 2)
             x = b_k;
       // Evaluate residual
         error = OBJECTIVE_Tscroll(x,Td,scroll,Inputs,UpstreamInputs);
       // First time through, store the outputs
         if(iter == 1)
       {
             f_ak = error;
             b_kn1 = a_k;
             f_bkn1 = error;
       }
         if (iter > 1)
       {
             f_bk = error;
          if (iter==2 && f_bk*f_bkn1>0)
          {
             if (f_bk<0)
             {
                printf_plus("In Dekker solver for Tscroll, point and contrapoint give
                    \n\t same sign of residual, moving \n\t the range down by 5K in
                    each direction\n");
                // Both guesses are too high, make the max value the old min, move
                    down the min value
                Ts_max=Ts_min;
                Ts_min=Ts_min-5;
             }
             else
             {
                printf_plus("In Dekker solver for Tscroll, point and contrapoint give
                    \n\t same sign of residual, moving \n\t the range up by 5K in each
                     direction\n");
```

```
                // Both guesses are too low, make the min value the old max, move up
                    the max value
                Ts_min=Ts_max;
                Ts_max=Ts_max+5;
            }
            iter=1;
            continue;
        }
        //Secant solution
            s = b_k - (b_k - b_kn1) / (f_bk - f_bkn1) * f_bk;
        //Midpoint solution
            m = (a_k + b_k) / 2.0;
            if (s > b_k && s < m)
        {
            b_kp1 = s; //Use the secant solution
            printf_plus("Secant step\n");
        }
        else
        {
                b_kp1 = m; //Use the midpoint solution
            printf_plus("Midpoint step\n");
        }
            // Evaluate objective function
            f_bkp1 = OBJECTIVE_Tscroll(b_kp1,Td,scroll,Inputs,UpstreamInputs);
        if (fabs(f_bkp1)<eps)
        {
            return b_kp1;
        }
        //See if the signs of iterate and contrapoint are the same
            if (f_ak*f_bkp1<0.0)
        {
                // a and b have opposite signs;
            // keep the same contrapoint
                a_kp1 = a_k;
                f_akp1 = f_ak;
        }
            else
        {
                //Otherwise, keep the iterate
                a_kp1 = b_k;
                f_akp1 = f_bk;
        }
            if (fabs(f_akp1) < fabs(f_bkp1))
        {
                //a_k+1 is a better guess than b_k+1, so swap a and b values for k+1
            swap(&a_kp1, &b_kp1);
                swap(&f_akp1, &f_bkp1);
        }
            //Update variables
            //Old values
            b_kn1 = b_k;
            f_bkn1 = f_bk;
            //values at this iterate
            b_k = b_kp1;
            a_k = a_kp1;
            f_ak = f_akp1;
            f_bk = f_bkp1;
        // resize the Td bounds for faster convergence
        if (fabs(a_k-b_k) > 2.0)
            scroll->solver.Td_halfBandWidth=2.0;
        else
            scroll->solver.Td_halfBandWidth=fabs(a_k-b_k);
      }
        iter++;
      if (iter>90 && fabs(error)>eps)
      {
        printf("Dekker for Tscroll has failed\n");
      }
   }
   //Finished step;
   OBJECTIVE_Tscroll(b_k,Td,scroll,Inputs,UpstreamInputs);
   printf_plus("%d Dekker steps taken for scroll loop",iter-1);
   return b_k;
}
double Secant_Tscroll(double Tscroll_guess,double Td, double delta, double eps,struct
     scrollVals *scroll, struct scrollInputVals *Inputs, struct scrollInputVals *
    UpstreamInputs)
{
   double x1,x2,x3,y1,y2,change,f,T;
```

```
    int iter;
    change=999;
    iter=1;
    while ((iter<=3 || change>eps) && iter<100)
    {
       if (iter==1){x1=Tscroll_guess; T=x1;}
       if (iter==2){x2=Tscroll_guess+delta; T=x2;}
       if (iter>2) {T=x2;}
          f=OBJECTIVE_Tscroll(T,Td,scroll,Inputs,UpstreamInputs);
       if (iter==1){y1=f;}
       if (iter>1)
       {
          y2=f;
          x3=x2-y2/(y2-y1)*(x2-x1);
          change=fabs(y2/(y2-y1)*(x2-x1));
          y1=y2; x1=x2; x2=x3;
       }
       iter=iter+1;
    }
    return T;
}
void MatInv_2(double A[2][2] , double B[2][2])
{
    double Det;
    //Using Cramer's Rule to invert 2x2 matrix
    Det=A[0][0]*A[1][1]-A[1][0]*A[0][1];
    B[0][0]=1.0/Det*A[1][1];
    B[1][1]=1.0/Det*A[0][0];
    B[1][0]=-1.0/Det*A[1][0];
    B[0][1]=-1.0/Det*A[0][1];
}
struct scrollVals Initialize_ScrollModel(struct geoVals *geo, struct scrollInputVals
     *Inputs, struct ExperVals *Exper, struct MLVals *ML, struct FlowVals *Flows)
{
    int i,iter=0,iterTd,LSCounter;
    struct scrollInputVals UpstreamInputs;
    struct scrollVals scroll;
    double mdot_suct,mdot_disc,mdot_th,hd_s,L_tube,D_tube,w;
    double Td_error=999, eps_Td=1e-4,Td_new,Td_min,Td_max,Tscroll;
    double P_gas, P_ML, h_disc,h_suct,mu_suct,mu_0,err_HT,s_suct;
    double m,b,c,Ls,Ms,Rs,Ld,Md,Rd,yLs,yMs,yRs,yLd,yMd,yRd;
    double kstar,Tlump,Td_ideal,Td;
    double x1,x2,x3,y1,y2,eps,change,f,T;
    double u1,u2,u3,v1,v2,epsTd,changeTd;
    double h1,h2,h2s,s1,T2bad,h2bad,TTsc,TTd;
    double Lw,Mw,Rw,yLw,yRw,yMw,objbase,tau,w1,w2,f1,f2;
    FILE *fp;
    clock_t t1,t2;
    double ybase[2],yplus1[2],yplus2[2],A[2][2],Ainv[2][2],x[2],dx[2],xold[2],yold[2],
        v[2];
    double Tout,mdot,T_scroll;
    t1=clock(); // Start timer;
    scroll.flags.lastRotation=false;
    LoadCVIndices(geo);
    // Everything in this function is executed once at the beginning of the scroll
        model
    // Allocate sufficient memory for a very large number of steps if needed
    // since the process of dynamic memory allocation is VERY slow in c
    scroll.geo=*geo;
    scroll.theta=     (double *)calloc(Ntheta_MAX,sizeof(double));
    scroll.V=         (double *)calloc(NCV*Ntheta_MAX,sizeof(double));
    scroll.dV=        (double *)calloc(NCV*Ntheta_MAX,sizeof(double));
    scroll.T=         (double *)calloc(NCV*Ntheta_MAX,sizeof(double));
    scroll.p=         (double *)calloc(NCV*Ntheta_MAX,sizeof(double));
    scroll.xL=        (double *)calloc(NCV*Ntheta_MAX,sizeof(double));
    scroll.rho=       (double *)calloc(NCV*Ntheta_MAX,sizeof(double));
    scroll.Q=         (double *)calloc(NCV*Ntheta_MAX,sizeof(double));
    scroll.m=         (double *)calloc(NCV*Ntheta_MAX,sizeof(double));
    scroll.HT.hc=     (double *)calloc(NCV*Ntheta_MAX,sizeof(double));
    scroll.HT.A_wall_i=  (double *)calloc(NCV*Ntheta_MAX,sizeof(double));
    scroll.HT.A_wall_o=  (double *)calloc(NCV*Ntheta_MAX,sizeof(double));
    scroll.HT.Tm_wall_i=(double *)calloc(NCV*Ntheta_MAX,sizeof(double));
    scroll.HT.Tm_wall_o=(double *)calloc(NCV*Ntheta_MAX,sizeof(double));
    scroll.HT.Tm_plate=  (double *)calloc(NCV*Ntheta_MAX,sizeof(double));
    scroll.Forces.Fx= (double *)calloc(NCV*Ntheta_MAX,sizeof(double));
    scroll.Forces.Fy= (double *)calloc(NCV*Ntheta_MAX,sizeof(double));
    scroll.Forces.Fz= (double *)calloc(NCV*Ntheta_MAX,sizeof(double));
    scroll.Forces.xcp=   (double *)calloc(NCV*Ntheta_MAX,sizeof(double));
```

```
scroll.Forces.ycp=   (double *)calloc(NCV*Ntheta_MAX,sizeof(double));
scroll.Forces.Mx= (double *)calloc(NCV*Ntheta_MAX,sizeof(double));
scroll.Forces.My= (double *)calloc(NCV*Ntheta_MAX,sizeof(double));
scroll.Forces.Mz= (double *)calloc(NCV*Ntheta_MAX,sizeof(double));
scroll.Forces.MO= (double *)calloc(NCV*Ntheta_MAX,sizeof(double));
scroll.error=  (double *)calloc(3*NCV*Ntheta_MAX,sizeof(double));
scroll.flowVec=   (struct flowVecVals*)malloc(Ntheta_MAX*sizeof(struct flowVecVals
    ));
scroll.N=Ntheta_MAX;
printf_plus("Memory allocated for scroll structure\n");

strcpy(scroll.Ref,Inputs->Ref);
strcpy(scroll.Liq,Inputs->Liq);
scroll.omega=Inputs->omega;

// Load up discharge port data for one revolution
for (i=0;i<NTHETA_ADISC;i++)
{
   scroll.geo.disc.thetaAdisc[i]=2*PI/(NTHETA_ADISC-1)*i;
   scroll.geo.disc.Adisc[i]=A_disc(geo,scroll.geo.disc.thetaAdisc[i]);
}
printf_plus("Discharge port area for one revolution calculated... \n");

// Copy Input structures
scroll.Exper=*Exper;
scroll.ML=*ML;
scroll.massFlow.Inputs=*Flows;

/* Assume the compression process to be an 70% efficient compression of
 oil-refrigerant mixture in order to get approximate discharge temp*/
h1=h_m(scroll.Ref,scroll.Liq,Inputs->T_in,Inputs->p_in,Inputs->xL_in);
s1=s_m(scroll.Ref,scroll.Liq,Inputs->T_in,Inputs->p_in,Inputs->xL_in);
h2s=h_sp(scroll.Ref,scroll.Liq,s1,Inputs->p_out,Inputs->xL_in,Inputs->T_in);
Td_ideal=T_hp(scroll.Ref,scroll.Liq,h2s,Inputs->p_out,Inputs->xL_in,Inputs->T_in);
h2=(h2s-h1)/scroll.ML.etac_guess+h1;
Inputs->T_out=T_hp(scroll.Ref,scroll.Liq,h2,Inputs->p_out,Inputs->xL_in,Inputs->
    T_in);
h2bad=(h2s-h1)/0.4+h1;
T2bad=T_hp(scroll.Ref,scroll.Liq,h2bad,Inputs->p_out,Inputs->xL_in,Inputs->T_in);

scroll.solver.Td_halfBandWidth=(Inputs->T_out-Inputs->T_in)/2.0;
printf_plus("Guess Td: %0.4f\n",Inputs->T_out);

//Save a backup copy of the inputs for the upstream location due to the HT in
    inlet port
UpstreamInputs=*Inputs;
scroll.Inputs=*Inputs;

//Make initial guess for mass flow based on 100% volumetric efficiency
mdot=Inputs->omega/(2.0*PI)*Vdisp(&(scroll.geo))*rho_m(scroll.Ref,scroll.Liq,
    Inputs->T_in,Inputs->p_in,Inputs->xL_in);
scroll.massFlow.mdot_tot=mdot;

// Ambient temperature
scroll.HT.T_amb=Inputs->T_amb;

Initialize_Rotation(&scroll,Inputs);
printf_plus("First rotation Initialized... \n\n");

//Td_min=0.5*(Td_ideal-Inputs->T_in)+Inputs->T_in;
//Td_max=0.75*(T2bad-Inputs->T_in)+Inputs->T_in;
//printf_plus("Hopefully Tscroll is in the range [%g,%g]\n",Td_min,Td_max);

// Actually run the solver for the scroll temperature
eps=1e-4;
//Secant_Tscroll(Td_min+30,Inputs->T_out,0.1,eps,&scroll,Inputs,&UpstreamInputs);
//Dekker_Tscroll(Td_min,Td_max,Td_min,eps,&scroll,Inputs,&UpstreamInputs);

// Block to do Newton-Raphson solver for discharge temp and lump temp
// Initial guess of the lump temperature is the outlet temperatures
Tlump=(Inputs->T_out+Inputs->T_in)/2.0;
x[0]=Tlump;
x[1]=Inputs->T_out;
dx[0]=1;
dx[1]=1;
ybase[0]=999;
ybase[1]=999;
w=1.0;
iter=0;
scroll.Debug.LumpHT_error_abs=999;
scroll.Debug.Td_error_abs=999;
// Use Newton-Raphson to get close to solution
while (fabs(scroll.Debug.LumpHT_error_abs)>0.0001 || fabs(scroll.Debug.
    Td_error_abs)>0.001)
{
   // ----------------------------------------
```

```
//              Newton-Raphson
// ---------------------------------------
//Base values
Tscroll=x[0];
Td=x[1];
OBJECTIVE_Tscroll(Tscroll,Td,&scroll,Inputs,&UpstreamInputs);
ybase[0]=scroll.Debug.LumpHT_error_abs;
ybase[1]=scroll.Debug.Td_error_abs;
objbase=sqrt(pow(scroll.Debug.LumpHT_error_abs,2)+pow(scroll.Debug.Td_error_abs
    ,2));
// Increment the lump temperature
Tscroll=x[0]+dx[0];
Td=x[1];
OBJECTIVE_Tscroll(Tscroll,Td,&scroll,Inputs,&UpstreamInputs);
yplus1[0]=scroll.Debug.LumpHT_error_abs;
yplus1[1]=scroll.Debug.Td_error_abs;
// Increment the discharge temperature
Tscroll=x[0];
Td=x[1]+dx[1];
OBJECTIVE_Tscroll(Tscroll,Td,&scroll,Inputs,&UpstreamInputs);
yplus2[0]=scroll.Debug.LumpHT_error_abs;
yplus2[1]=scroll.Debug.Td_error_abs;
//Build Jacobian matrix using numerical derivatives
A[0][0]=(yplus1[0]-ybase[0])/dx[0];
A[0][1]=(yplus1[1]-ybase[1])/dx[0];
A[1][0]=(yplus2[0]-ybase[0])/dx[1];
A[1][1]=(yplus2[1]-ybase[1])/dx[1];
//Invert matrix
MatInv_2(A,Ainv);
// Search direction
v[0]=-w*(Ainv[0][0]*ybase[0]+Ainv[1][0]*ybase[1]);
v[1]=-w*(Ainv[0][1]*ybase[0]+Ainv[1][1]*ybase[1]);
w=1.0;
if (iter>10 && iter%5==0)
{
   // ------------------------------------------------
   //                 Line Search
   // ------------------------------------------------
   // Line search along search direction given by NR
   Lw=0.1;
   Rw=3.0;
   yLw=objbase;
   Tscroll=x[0];
   Td=x[1];
   // Try the right bound
   OBJECTIVE_Tscroll(Tscroll+Rw*v[0],Td+Rw*v[1],&scroll,Inputs,&UpstreamInputs)
       ;
   yRw=sqrt(pow(scroll.Debug.LumpHT_error_abs,2)+pow(scroll.Debug.Td_error_abs
       ,2));
   tau=(sqrt(5.0)-1.0)/2.0;
   LSCounter=0;
   // Internal step 1
   w1=Lw+(1-tau)*(Rw-Lw);
   OBJECTIVE_Tscroll(Tscroll+w1*v[0],Td+w1*v[1],&scroll,Inputs,&UpstreamInputs)
       ;
   f1=sqrt(pow(scroll.Debug.LumpHT_error_abs,2)+pow(scroll.Debug.Td_error_abs
       ,2));
   // Internal step 2
   w2=Lw+tau*(Rw-Lw);
   OBJECTIVE_Tscroll(Tscroll+w2*v[0],Td+w2*v[1],&scroll,Inputs,&UpstreamInputs)
       ;
   f2=sqrt(pow(scroll.Debug.LumpHT_error_abs,2)+pow(scroll.Debug.Td_error_abs
       ,2));
   // If both of the internal functional values are less than both bounds, do a
        golden section line search
   if ((f2<yLw || f2<yRw) && (f1<yRw || f1<yLw))
   {
      while (LSCounter<4 && fabs(w2-w1)>0.01)
      {
         printf_plus("Line search, functional values are %g %g %g %g\n",yLw,f1,
             f2,yRw);
         printf_plus("Line search, values are %g %g %g %g\n",Lw,w1,w2,Rw);
         if (f1>f2)
         {
            w=w2; // new step size
            Lw=w1;
            w1=w2;
            f1=f2;
            w2=Lw+tau*(Rw-Lw);
```

```
                    OBJECTIVE_Tscroll(Tscroll+w2*v[0],Td+w2*v[1],&scroll,Inputs,&
                        UpstreamInputs);
                    f2=sqrt(pow(scroll.Debug.LumpHT_error_abs,2)+pow(scroll.Debug.
                        Td_error_abs,2));
                }
                else
                {
                    w=w1; // new step size
                    Rw=w2;
                    w2=w1;
                    f2=f1;
                    w1=Lw+(1-tau)*(Rw-Lw);
                    OBJECTIVE_Tscroll(Tscroll+w1*v[0],Td+w1*v[1],&scroll,Inputs,&
                        UpstreamInputs);
                    f1=sqrt(pow(scroll.Debug.LumpHT_error_abs,2)+pow(scroll.Debug.
                        Td_error_abs,2));
                }
                LSCounter++;
            }
        }
        // Otherwise use the minimum of the bound values
        else
        {
            printf_plus("Line search didn't improve results, values are %g %g %g %g\n
                ",yLw,f1,f2,yRw);
            if (yLw<yRw)
                w=1.0;
            else if (yRw<yLw)
                w=Rw;
        }
    }
    // Get new values
    x[0]=x[0]-w*(Ainv[0][0]*ybase[0]+Ainv[1][0]*ybase[1]);
    x[1]=x[1]-w*(Ainv[0][1]*ybase[0]+Ainv[1][1]*ybase[1]);
    // Final run with finished values
    Tscroll=x[0];
    Td=x[1];
    OBJECTIVE_Tscroll(Tscroll,Td,&scroll,Inputs,&UpstreamInputs);
    // Increment counter
    iter++;
}
h_disc=h_m(scroll.Ref,scroll.Liq,scroll.T[Idischarge],scroll.p[Idischarge],scroll.
    xL[Idischarge]);
h_suct=h_m(scroll.Ref,scroll.Liq,scroll.T[Isuction],scroll.p[Isuction],scroll.xL[
    Isuction]);
scroll.PowerEff.P_shaft=scroll.massFlow.mdot_tot*(h_disc-h_suct)-scroll.HT.
    Q_scroll_amb;
mdot_th=Inputs->omega/(2.0*PI)*Vdisp(&(scroll.geo))*rho_m(scroll.Ref,scroll.Liq,
    Inputs->T_in,Inputs->p_in,Inputs->xL_in);
scroll.PowerEff.eta_v=scroll.massFlow.mdot_tot/mdot_th;
s_suct=s_m(scroll.Ref,scroll.Liq,UpstreamInputs.T_in,UpstreamInputs.p_in,
    UpstreamInputs.xL_in);
hd_s=h_sp(scroll.Ref,scroll.Liq,s_suct,scroll.p[Idischarge],scroll.xL[Idischarge],
    scroll.T[Idischarge]);
scroll.PowerEff.eta_c=scroll.massFlow.mdot_tot*(hd_s-h_suct)/scroll.PowerEff.
    P_shaft;
scroll.PowerEff.eta_m = scroll.PowerEff.P_gas / scroll.PowerEff.P_shaft;
// Print and calculate a couple of things for debug purposes
printf_plus("mdots: %g mdotd: %g Error[%%]: %0.3f\n",scroll.massFlow.mdot_suct,
    scroll.massFlow.mdot_disc,(fabs(scroll.massFlow.mdot_suct)-fabs(scroll.
    massFlow.mdot_disc))/fabs(scroll.massFlow.mdot_disc)*100.0);
printf_plus("eta_v: %g\n",scroll.massFlow.mdot_suct/(Inputs->omega/(2.0*PI)*Vdisp
    (&(scroll.geo))*rho_m(scroll.Ref,scroll.Liq,Inputs->T_in,Inputs->p_in,Inputs->
    xL_in)));
printf_plus("eta_i: %g\n",scroll.PowerEff.eta_c);
scroll.Debug.mdot_error_abs=fabs(scroll.massFlow.mdot_suct)-fabs(scroll.massFlow.
    mdot_disc);
scroll.Debug.Ntheta=Ntheta;
t2=clock();
scroll.Debug.ElapsedTime= ((double)(t2-t1))/CLOCKS_PER_SEC/60.0; // Elapsed time
    for this run in minutes
//Post-Processing
CalculateLossTerms(&scroll);
CalculateForces(&scroll,Ntheta);
saveOutputs(&scroll);
return scroll;
}
```

```
void Rotation_RK45(struct scrollVals *scroll, struct scrollInputVals *Inputs)
{
   /* This function implements an adaptive Runge-Kutta-Feldberg 5th order
   solver for the system of equations of temperature, pressure and oil mass
   fraction.
   Mathematically this can be expressed as:
   k1=h*dy(xn                                                              ,t);
    k2=h*dy(xn+1.0/4.0*k1                                                   ,t
        +1.0/4.0*h);
    k3=h*dy(xn+3.0/32.0*k1+9.0/32.0*k2                                      ,t
        +3.0/8.0*h);
    k4=h*dy(xn+1932.0/2197.0*k1-7200.0/2197.0*k2+7296.0/2197.0*k3           ,t
        +12.0/13.0*h);
    k5=h*dy(xn+439.0/216.0*k1-8.0*k2+3680.0/513.0*k3-845.0/4104.0*k4        ,t+h
        );
    k6=h*dy(xn-8.0/27.0*k1+2.0*k2-3544.0/2565.0*k3+1859.0/4104.0*k4-11.0/40.0*k5 ,t
        +1.0/2.0*h);
   where the function dy(y,t) returns a vector of the ODE expressions.
   The new value is calculated from
   xnplus=xn+gamma1*k1+gamma2*k2+gamma3*k3+gamma4*k4+gamma5*k5+gamma6*k6
   In the adaptive solver, the errors for a given step can be calculated from
   error=1.0/360.0*k1-128.0/4275.0*k3-2197.0/75240.0*k4+1.0/50.0*k5+2.0/55.0*k6;
   If the maximum absolute error is above 1.0e-5, the step size is halved and the
       step is
   tried again until the error is below 1.0e-5.  If the error is less than 1.0e-7,
       the step
   size is doubled to minimize the number of steps required.
   Three special cases are important to handle:
   1) At the discharge angle, turn off adaptive step-sizing and use the previous
       angle
      to simplify the calculations.
   2) At the merging point, turn off adaptive step-sizing.
   3) At basically the ending angle, take a baby step and turn off adaptive sizing.
   */
   int i,j,Itheta,disableAdaptive,nextStepDischarge,nRot=0,lastRotation=false;
   int num=0,Ierror=0;
   double *f1,*f2,*f3,*f4,*f5,*f6,*error,h,t0;
   double *var1,*var2,*var3;
   double p_d1,p_d2,p_dd;
   double gamma1=16.0/135.0;
    double gamma2=0.0;
    double gamma3=6656.0/12825.0;
    double gamma4=28561.0/56430.0;
    double gamma5=-9.0/50.0;
    double gamma6=2.0/55.0;
   double eps_max,eps_min;
   double wrap_error,eps_wrap;
   double max_error=999.0;
   clock_t t1,t2;
   wrap_error=999;
   /* Clear all the flowVec to ensure no old values
   remain from a prior rotation */
   for (i=0;i<Ntheta;i++)
   {
      freeFlowVec(&(scroll->flowVec[i]));
   }
   /* Flush out any old values from a prior rotation */
   for (i=1;i<Ntheta+1000;i++) //+1000 to make sure that any stragglers due to
       resizing are also cleared
   {
      for (j=0;j<NCV;j++)
      {
         scroll->dV[j+i*NCV]=0.0;       scroll->V[j+i*NCV]=0.0;
         scroll->T[j+i*NCV]=0.0;        scroll->p[j+i*NCV]=0.0;
         scroll->xL[j+i*NCV]=0.0;       scroll->Q[j+i*NCV]=0.0;
         scroll->m[j+i*NCV]=0.0;
         scroll->HT.hc[j+i*NCV]=0.0;
         scroll->HT.A_wall_i[j+i*NCV]=0.0;
         scroll->HT.A_wall_o[j+i*NCV]=0.0;
         scroll->HT.Tm_wall_i[j+i*NCV]=0.0;
         scroll->HT.Tm_wall_o[j+i*NCV]=0.0;
         scroll->HT.Tm_plate[j+i*NCV]=0.0;
      }
   }
   // Reset the discharge temp in case it is the first call
   //   of a rotation block
```

```
scroll->T[Idischarge]=Inputs->T_out;
// Depending on what state variables are chosen, make pointers
// var1, var2, and var3 point to the memory locations of the arrays
// for the variables needed
if (ODEVars==T_p_xL)
{
   var1=scroll->T;
   var2=scroll->p;
   var3=scroll->xL;
}
if (ODEVars==T_m_xL)
{
   var1=scroll->T;
   var2=scroll->m;
   var3=scroll->xL;
}
do
{
   dtCounter=0;
   dTmmCounter=0;
   Itheta=0;
   Ntheta=0;
   t0=0.0;
   h=0.00001; //Initial step size at theta=0.0
   nextStepDischarge=false;
   eps_max=1e-6;
   t1=clock();
   while (t0<2*PI)
   {
      disableAdaptive=false;
      //printf_plus("t: %0.8f \t\t %0.8f \t\t %g\n",t0,scroll->xL[Is1+NCV*(Itheta)
          ],theta_d(&(scroll->geo)));
      scroll->theta[Itheta]=t0;
      while (max_error>eps_max && disableAdaptive==false)
      {
      scroll->theta[Itheta+1]=t0+h;
      // If t0 and t0+h would bracket the discharge angle,
      // take a small step to get you just short of the discharge angle,
      // then the next step reassign compression and discharge chambers
      if ( t0 < theta_d(&(scroll->geo)) && (t0+h) > theta_d(&(scroll->geo))
         && scroll->flags.LeftDischarge==false && nextStepDischarge==false)
      {
         h=theta_d(&(scroll->geo))-t0-1e-6;
         scroll->flags.LeftDischarge=false;
         disableAdaptive=true;
         nextStepDischarge=true; // The next step will go into the discharge
             region
      }
      else if (nextStepDischarge==true)
      {
         // Adaptive makes steps of h/4 3h/8 12h/13 and h/2 and h
         // Make sure step does not hit any *right* at theta_d
         // That is why it is 2.2 rather than 2.0
         h=2.2e-6;
         AtDischarge(scroll,Itheta,t0);
         scroll->flags.LeftDischarge=true;
         scroll->flags.useDDD=false;
         disableAdaptive=true;
         nextStepDischarge=false;
      }
      else
      {
         scroll->flags.LeftDischarge=false;
         nextStepDischarge=false;
      }
      /* If you are at the end of the rotation, and a step at a size of
      h would result in going past 2*pi, take a fractional step so you
      end up at 2*pi and turn off adaptive sizing */
      if (t0+h>2*PI)
      {
         h=2*PI-t0;
         disableAdaptive=true;
      }
      /* If the d1-d2-dd chambers have equalized in pressure to
      within 0.05% of each other, merge the chambers together by
      averaging their properties.  This only occurs after the
      discharge angle
      If the d1 or d2 chamber is higher in pressure than the dd chamber,
```

```
the change in volume of the two chambers do not tend to drive
the chamber pressures together, so it is necessary to use a much more
generous convergence criterion for the merge of 2.0%
*/
if (scroll->flags.useDDD==0)
{
   // Define terms for compactness
   p_d1=scroll->p[Id1+NCV*Itheta];
   p_d2=scroll->p[Id2+NCV*Itheta];
   p_dd=scroll->p[Idd+NCV*Itheta];
   if ( fabs((p_d1-p_dd)/p_dd)<0.0005 || ( (p_d2 > p_dd) && fabs((p_d2-p_dd)
       /p_dd)<0.02 ) || scroll->dV[Idd+NCV*Itheta]>0 )
   {
      // Make sure you aren't too close to the discharge angle so that more
          than
      // one special thing is happening at the step (merge,split,discharge)
      if ( fabs(t0-theta_d(&(scroll->geo)))>0.1)
      {
         Merge(scroll,Itheta,t0);
      }
   }
}
error=(double *)calloc(NCV*3,sizeof(double));
// Step 1: derivatives evaluated at previous step
f1=Derivs(scroll,Itheta,t0);
////Update temporary storage values in the next column of the matrix
for (i=0;i<NCV;i++)
{
   var1[i+NCV*(Itheta+1)]     =var1[i+NCV*Itheta]    +h*(+1.0/4.0*f1[i]);
   var2[i+NCV*(Itheta+1)]     =var2[i+NCV*Itheta]    +h*(+1.0/4.0*f1[i+NCV]);
   var3[i+NCV*(Itheta+1)]     =var3[i+NCV*Itheta]   +h*(+1.0/4.0*f1[i+2*NCV])
       ;
}
f2=Derivs(scroll,Itheta+1,t0+1.0/4.0*h);
freeFlowVec(&(scroll->flowVec[Itheta+1]));
////Update temporary storage values in the next column of the matrix
for (i=0;i<NCV;i++)
{
   var1[i+NCV*(Itheta+1)]     =var1[i+NCV*Itheta]    +h*(+3.0/32.0*f1[i]
           +9.0/32.0*f2[i]);
   var2[i+NCV*(Itheta+1)]     =var2[i+NCV*Itheta]    +h*(+3.0/32.0*f1[i+NCV]
        +9.0/32.0*f2[i+NCV]);
   var3[i+NCV*(Itheta+1)]     =var3[i+NCV*Itheta]   +h*(+3.0/32.0*f1[i+2*NCV
       ]+9.0/32.0*f2[i+2*NCV]);
}
f3=Derivs(scroll,Itheta+1,t0+3.0/8.0*h);
freeFlowVec(&(scroll->flowVec[Itheta+1]));
////Update temporary storage values in the next column of the matrix
for (i=0;i<NCV;i++)
{
   var1[i+NCV*(Itheta+1)]     =var1[i+NCV*Itheta]    +h*(+1932.0/2197.0*f1[i]
           -7200.0/2197.0*f2[i]       +7296.0/2197.0*f3[i]);
   var2[i+NCV*(Itheta+1)]     =var2[i+NCV*Itheta]    +h*(+1932.0/2197.0*f1[i+
       NCV]   -7200.0/2197.0*f2[i+NCV]   +7296.0/2197.0*f3[i+NCV]);
   var3[i+NCV*(Itheta+1)]     =var3[i+NCV*Itheta]   +h*(+1932.0/2197.0*f1[i
       +2*NCV]-7200.0/2197.0*f2[i+2*NCV]+7296.0/2197.0*f3[i+2*NCV]);
}
f4=Derivs(scroll,Itheta+1,t0+12.0/13.0*h);
freeFlowVec(&(scroll->flowVec[Itheta+1]));
////Update temporary storage values in the next column of the matrix
for (i=0;i<NCV;i++)
{
   var1[i+NCV*(Itheta+1)]     =var1[i+NCV*Itheta]    +h*(+439.0/216.0*f1[i]
           -8.0*f2[i]       +3680.0/513.0*f3[i]       -845.0/4104.0*f4[i]);
   var2[i+NCV*(Itheta+1)]     =var2[i+NCV*Itheta]    +h*(+439.0/216.0*f1[i+
       NCV]   -8.0*f2[i+NCV]   +3680.0/513.0*f3[i+NCV]   -845.0/4104.0*f4[i+NCV
       ]);
   var3[i+NCV*(Itheta+1)]     =var3[i+NCV*Itheta]   +h*(+439.0/216.0*f1[i+2*
       NCV]-8.0*f2[i+2*NCV]+3680.0/513.0*f3[i+2*NCV]-845.0/4104.0*f4[i+2*NCV
       ]);
}
f5=Derivs(scroll,Itheta+1,t0+h);
freeFlowVec(&(scroll->flowVec[Itheta+1]));
////Update temporary storage values in the next column of the matrix
for (i=0;i<NCV;i++)
{
   var1[i+NCV*(Itheta+1)]     =var1[i+NCV*Itheta]    +h*(-8.0/27.0*f1[i]
           +2.0*f2[i]       -3544.0/2565.0*f3[i]       +1859.0/4104.0*f4[i]
           -11.0/40.0*f5[i]);
```

```
        var2[i+NCV*(Itheta+1)]     =var2[i+NCV*Itheta]   +h*(-8.0/27.0*f1[i+NCV]
             +2.0*f2[i+NCV]  -3544.0/2565.0*f3[i+NCV]  +1859.0/4104.0*f4[i+NCV]
            -11.0/40.0*f5[i+NCV]);
        var3[i+NCV*(Itheta+1)]     =var3[i+NCV*Itheta]   +h*(-8.0/27.0*f1[i+2*NCV
            ]+2.0*f2[i+2*NCV]-3544.0/2565.0*f3[i+2*NCV]+1859.0/4104.0*f4[i+2*NCV
            ]-11.0/40.0*f5[i+2*NCV]);
    }
    f6=Derivs(scroll,Itheta+1,t0+1.0/2.0*h);
    freeFlowVec(&(scroll->flowVec[Itheta+1]));
    /* Updated values after the step */
    for (i=0;i<NCV;i++)
    {
        var1[i+NCV*(Itheta+1)]     =var1[i+NCV*Itheta]   +h*(+gamma1*f1[i]      +
            gamma2*f2[i]      +gamma3*f3[i]      +gamma4*f4[i]      +gamma5*f5[i]
                +gamma6*f6[i]);
        var2[i+NCV*(Itheta+1)]     =var2[i+NCV*Itheta]   +h*(+gamma1*f1[i+NCV]  +
            gamma2*f2[i+NCV]  +gamma3*f3[i+NCV]  +gamma4*f4[i+NCV]  +gamma5*f5[i+
            NCV]  +gamma6*f6[i+NCV]);
        var3[i+NCV*(Itheta+1)]     =var3[i+NCV*Itheta]  +h*(+gamma1*f1[i+2*NCV]+
            gamma2*f2[i+2*NCV]+gamma3*f3[i+2*NCV]+gamma4*f4[i+2*NCV]+gamma5*f5[i
            +2*NCV]+gamma6*f6[i+2*NCV]);
    }
    max_error=0.0;
    Ierror=0;
    for (i=0;i<3*NCV;i++)
    {
        error[i]=h*(1.0/360.0*f1[i]-128.0/4275.0*f3[i]-2197.0/75240.0*f4[i
            ]+1.0/50.0*f5[i]+2.0/55.0*f6[i]);
        if (fabs(error[i])>max_error)
        {
            max_error=fabs(error[i]);
            Ierror=i;
        }
        scroll->error[i+3*NCV*Itheta]=error[i];
    }
    free(error);
    //printf_plus("theta: %g \t Max error:%g \t Ierror: %d nC: %d\n",t0+h,
        max_error,(Ierror+1)%NCV,nC(&(scroll->geo),t0));
    free(f1); free(f2); free(f3); free(f4); free(f5); free(f6);
    /* If the error is too large, take a smaller step next time*/
    if (max_error>eps_max && disableAdaptive==false)
    {
        h*=0.9*pow(eps_max/max_error,0.3);
        /* Free the old flowVec */
        freeFlowVec(&(scroll->flowVec[Itheta]));
    }
}
t0=t0+h;
Itheta++;
if (Itheta==4480000)
{
    Ntheta=Itheta+1;
    Matrix2csv("../../theta.csv",scroll->theta,1,Ntheta);
    Matrix2csv("../../T.csv",scroll->T,NCV,Ntheta);
    Matrix2csv("../../p.csv",scroll->p,NCV,Ntheta);
    Matrix2csv("../../xL.csv",scroll->xL,NCV,Ntheta);
    Matrix2csv("../../V.csv",scroll->V,NCV,Ntheta);
    Matrix2csv("../../dV.csv",scroll->dV,NCV,Ntheta);
    Matrix2csv("../../Q.csv",scroll->Q,NCV,Ntheta);
    Matrix2csv("../../hc.csv",scroll->HT.hc,NCV,Ntheta);
    printf("theta %g thetad: %g\n",t0,theta_d(&(scroll->geo)));
    Ntheta=Ntheta-1;
}
/* If the error is very small, take a bigger step next time*/
if (max_error<eps_max && disableAdaptive==false)
{
    h *= 0.9*pow(eps_max/max_error,0.2);
}
/* Reset error terms */
max_error=999.0;
disableAdaptive=false;
}
/*Evaluate functions after the step */
f6=Derivs(scroll,Itheta,t0);
scroll->theta[Itheta]=t0;
free(f6);
Ntheta=Itheta+1;
scroll->Ntheta=Ntheta;
```

```
        t2=clock();
        printf_plus("\t\tTime for one rotation(): %g [s] \n",((double)(t2-t1))/
            CLOCKS_PER_SEC);
        Ntheta=Itheta+1;
        wrap_error=Wrap_Error(scroll);
        printf_plus("\t\tWrap_error: %g %% \n",wrap_error*100.0);
        eps_wrap=0.00001;
        ///* The max error per step is 0.001% */
        if (wrap_error> eps_wrap )//|| nRot==0)
        {
           /* Connect beginning of the last rotation with the new rotation
           (pass old values at the end of the rotation to new values) */
           Wrap_Rotation(scroll,Inputs);
           printf_plus("\t\tWrapping\n");
           wrap_error=999;
        }
        nRot++;
        //printf_plus("dT_dp_dxL evaluated %d times for %d steps or average of %g calls
            /step\n",dTmmCounter,Ntheta, (double)dTmmCounter/((double)Ntheta) );
     }
     while(wrap_error>eps_wrap || (wrap_error>10.0*eps_wrap && nRot<10) );
     scroll->Debug.wrap_error_rel=wrap_error;
}
void Wrap_Rotation(struct scrollVals *scroll, struct scrollInputVals *Inputs)
{
     int i,j;
     /* Clear out the first column of data */
     for (i=0;i<NCV;i++)
     {
        scroll->T[i]=0.0; scroll->p[i]=0.0; scroll->xL[i]=0.0;
     }
     scroll->T[Isa]=scroll->T[Isa+NCV*(Ntheta-1)];
     scroll->p[Isa]=scroll->p[Isa+NCV*(Ntheta-1)];
     scroll->xL[Isa]=scroll->xL[Isa+NCV*(Ntheta-1)];
     scroll->T[Is1]=Inputs->T_in;
     scroll->p[Is1]=Inputs->p_in;
     scroll->xL[Is1]=Inputs->xL_in;
     scroll->T[Is2]=Inputs->T_in;
     scroll->p[Is2]=Inputs->p_in;
     scroll->xL[Is2]=Inputs->xL_in;
     scroll->T[Isuction]=Inputs->T_in;
     scroll->p[Isuction]=Inputs->p_in;
     scroll->xL[Isuction]=Inputs->xL_in;
     scroll->T[Idischarge]=Inputs->T_out;
     scroll->p[Idischarge]=Inputs->p_out;
     scroll->xL[Idischarge]=Inputs->xL_in;
     scroll->T[Ic1[0]]=scroll->T[Is1+NCV*(Ntheta-1)];
     scroll->p[Ic1[0]]=scroll->p[Is1+NCV*(Ntheta-1)];
     scroll->xL[Ic1[0]]=scroll->xL[Is1+NCV*(Ntheta-1)];
     scroll->T[Ic2[0]]=scroll->T[Is2+NCV*(Ntheta-1)];
     scroll->p[Ic2[0]]=scroll->p[Is2+NCV*(Ntheta-1)];
     scroll->xL[Ic2[0]]=scroll->xL[Is2+NCV*(Ntheta-1)];
     for (j=1;j<nC_Max(&(scroll->geo));j++)
     {
        scroll->T[Ic1[j]]=scroll->T[Ic1[j-1]+NCV*(Ntheta-1)];
        scroll->p[Ic1[j]]=scroll->p[Ic1[j-1]+NCV*(Ntheta-1)];
        scroll->xL[Ic1[j]]=scroll->xL[Ic1[j-1]+NCV*(Ntheta-1)];
        scroll->T[Ic2[j]]=scroll->T[Ic2[j-1]+NCV*(Ntheta-1)];
        scroll->p[Ic2[j]]=scroll->p[Ic2[j-1]+NCV*(Ntheta-1)];
        scroll->xL[Ic2[j]]=scroll->xL[Ic2[j-1]+NCV*(Ntheta-1)];
     }
     if (scroll->flags.useDDD==true)
     {
        scroll->T[Iddd]=scroll->T[Iddd+NCV*(Ntheta-1)];
        scroll->p[Iddd]=scroll->p[Iddd+NCV*(Ntheta-1)];
        scroll->xL[Iddd]=scroll->xL[Iddd+NCV*(Ntheta-1)];
     }
     else
     {
        scroll->T[Id1]=scroll->T[Id1+NCV*(Ntheta-1)];
        scroll->p[Id1]=scroll->p[Id1+NCV*(Ntheta-1)];
        scroll->xL[Id1]=scroll->xL[Id1+NCV*(Ntheta-1)];
        scroll->T[Id2]=scroll->T[Id2+NCV*(Ntheta-1)];
        scroll->p[Id2]=scroll->p[Id2+NCV*(Ntheta-1)];
```

```
      scroll->xL[Id2]=scroll->xL[Id2+NCV*(Ntheta-1)];
      scroll->T[Idd]=scroll->T[Idd+NCV*(Ntheta-1)];
      scroll->p[Idd]=scroll->p[Idd+NCV*(Ntheta-1)];
      scroll->xL[Idd]=scroll->xL[Idd+NCV*(Ntheta-1)];
   }
   /* Clear out the rest of the matrix */
   for (i=0;i<NCV;i++)
   {
      /* Start at the second column since the first column is
      the new data which should be preserved */
      for (j=1;j<Ntheta;j++)
      {
         scroll->T[i+NCV*j]=0.0;
         scroll->p[i+NCV*j]=0.0;
         scroll->xL[i+NCV*j]=0.0;
         scroll->V[i+NCV*j]=0.0;
         scroll->dV[i+NCV*j]=0.0;
         scroll->Q[i+NCV*j]=0.0;
         scroll->HT.hc[i+NCV*j]=0.0;
         scroll->Forces.Fx[i+NCV*j]=0.0;
         scroll->Forces.Fy[i+NCV*j]=0.0;
         scroll->Forces.Fz[i+NCV*j]=0.0;
         scroll->Forces.Mx[i+NCV*j]=0.0;
         scroll->Forces.My[i+NCV*j]=0.0;
         scroll->Forces.Mz[i+NCV*j]=0.0;
         scroll->Forces.xcp[i+NCV*j]=0.0;
         scroll->Forces.ycp[i+NCV*j]=0.0;
      }
   }
   for (j=0;j<Ntheta;j++)
   {
      freeFlowVec(&(scroll->flowVec[j]));
   }
}
double Wrap_Error(struct scrollVals *scroll)
{
   double max_error=0.0,P1,P2;
   int j;
   if (scroll->flags.useDDD)
   {
      P1=scroll->p[Iddd]; P2=scroll->p[Iddd+NCV*(Ntheta-1)];
      if (fabs((P1-P2)/P1)>max_error)
         max_error=fabs((P1-P2)/P1);
   }
   else
   {
      P1=scroll->p[Id1]; P2=scroll->p[Id1+NCV*(Ntheta-1)];
      if (fabs((P1-P2)/P1)>max_error)
         max_error=fabs((P1-P2)/P1);
      P1=scroll->p[Id2]; P2=scroll->p[Id2+NCV*(Ntheta-1)];
      if (fabs((P1-P2)/P1)>max_error)
         max_error=fabs((P1-P2)/P1);
      P1=scroll->p[Idd]; P2=scroll->p[Idd+NCV*(Ntheta-1)];
      if (fabs((P1-P2)/P1)>max_error)
         max_error=fabs((P1-P2)/P1);
   }
   P1=scroll->p[Ic1[0]]; P2=scroll->p[Is1+NCV*(Ntheta-1)];
   if (fabs((P1-P2)/P1)>max_error)
      max_error=fabs((P1-P2)/P1);
   P1=scroll->p[Ic2[0]]; P2=scroll->p[Is2+NCV*(Ntheta-1)];
   if (fabs((P1-P2)/P1)>max_error)
      max_error=fabs((P1-P2)/P1);
   for (j=1;j<nC_Max(&(scroll->geo));j++)
   {
      P1=scroll->p[Ic1[j]]; P2=scroll->p[Ic1[j-1]+NCV*(Ntheta-1)];
      if (fabs((P1-P2)/P1)>max_error)
         max_error=fabs((P1-P2)/P1);
      P1=scroll->p[Ic2[j]]; P2=scroll->p[Ic2[j-1]+NCV*(Ntheta-1)];
      if (fabs((P1-P2)/P1)>max_error)
         max_error=fabs((P1-P2)/P1);
   }
   return max_error;
}
void Merge(struct scrollVals *scroll, int Itheta,double theta)
{
   double rho_d1,rho_d2,rho_dd,m_oil,m_ref,u_dd,u_d1,u_d2,U_ddd,rho_ddd,m_ddd,u_ddd;
```

```
    double V_d1,T_d1,p_d1,xL_d1,V_d2,T_d2,p_d2,xL_d2,V_dd,T_dd,p_dd,xL_dd;
    scroll->flags.useDDD=1;
    V_d1=scroll->V[Id1+NCV*Itheta];
    T_d1=scroll->T[Id1+NCV*Itheta];
    p_d1=scroll->p[Id1+NCV*Itheta];
    xL_d1=scroll->xL[Id1+NCV*Itheta];
    V_d2=scroll->V[Id2+NCV*Itheta];
    T_d2=scroll->T[Id2+NCV*Itheta];
    p_d2=scroll->p[Id2+NCV*Itheta];
    xL_d2=scroll->xL[Id2+NCV*Itheta];
    V_dd=scroll->V[Idd+NCV*Itheta];
    T_dd=scroll->T[Idd+NCV*Itheta];
    p_dd=scroll->p[Idd+NCV*Itheta];
    xL_dd=scroll->xL[Idd+NCV*Itheta];
    rho_d1=rho_m(scroll->Ref,scroll->Liq,T_d1,p_d1,xL_d1);
     rho_d2=rho_m(scroll->Ref,scroll->Liq,T_d2,p_d2,xL_d2);
     rho_dd=rho_m(scroll->Ref,scroll->Liq,T_dd,p_dd,xL_dd);
    u_d1=u_m(scroll->Ref,scroll->Liq,T_d1,p_d1,xL_d1);
     u_d2=u_m(scroll->Ref,scroll->Liq,T_d2,p_d2,xL_d2);
     u_dd=u_m(scroll->Ref,scroll->Liq,T_dd,p_dd,xL_dd);
    scroll->p[Iddd+NCV*Itheta]=(V_dd*p_dd+V_d1*p_d1+V_d2*p_d2)/(V_d1+V_d2+V_dd);
    /* Mass fraction of mixed chamber is
      m_oil(total)/(m_oil(total)+m_gas(total))*/
    m_oil=V_dd*rho_dd*xL_dd
        +V_d1*rho_d1*xL_d1
        +V_d2*rho_d2*xL_d2;
    m_ref=V_dd*rho_dd*(1-xL_dd)
        +V_d1*rho_d1*(1-xL_d1)
        +V_d2*rho_d2*(1-xL_d2);
    scroll->xL[Iddd+NCV*Itheta]=m_oil/(m_ref+m_oil);
    U_ddd=V_dd*rho_dd*u_dd+V_d1*rho_d1*u_d1+V_d2*rho_d2*u_d2;
    scroll->T[Iddd+NCV*Itheta]=(V_dd*T_dd+V_d1*T_d1+V_d2*T_d2)/(V_d1+V_d2+V_dd);
    // Doesn't seem to work using energy balance unfortunately
    //scroll->T[Iddd+NCV*Itheta]=T_Up(scroll->Ref,scroll->Liq,U_ddd,scroll->p[Iddd+NCV
        *Itheta],scroll->xL[Iddd+NCV*Itheta],V_dd+V_d1+V_d2,T_d1);
    scroll->m[Iddd+NCV*Itheta]=m_oil+m_ref;
    scroll->T[Id1+NCV*Itheta]=0.0; scroll->p[Id1+NCV*Itheta]=0.0;
    scroll->xL[Id1+NCV*Itheta]=0.0; scroll->m[Id1+NCV*Itheta]=0.0;
    scroll->V[Id1+NCV*Itheta]=0.0; scroll->dV[Id1+NCV*Itheta]=0.0;
    scroll->T[Id2+NCV*Itheta]=0.0; scroll->p[Id2+NCV*Itheta]=0.0;
    scroll->xL[Id2+NCV*Itheta]=0.0; scroll->m[Id2+NCV*Itheta]=0.0;
    scroll->V[Id2+NCV*Itheta]=0.0; scroll->dV[Id2+NCV*Itheta]=0.0;
    scroll->T[Idd+NCV*Itheta]=0.0; scroll->p[Idd+NCV*Itheta]=0.0;
    scroll->xL[Idd+NCV*Itheta]=0.0; scroll->m[Idd+NCV*Itheta]=0.0;
    scroll->V[Idd+NCV*Itheta]=0.0; scroll->dV[Idd+NCV*Itheta]=0.0;
}
void AtDischarge(struct scrollVals *scroll, int Itheta,double theta)
{
    int Nc;
    // In this function Itheta is the index to the left of the discharge angle;
    scroll->flags.useDDD=0;
    Nc=nC(&(scroll->geo),theta);
    scroll->T[Idd+NCV*Itheta]=scroll->T[Iddd+NCV*Itheta];
    scroll->T[Id1+NCV*Itheta]=scroll->T[Ic1[Nc-1]+NCV*Itheta];
    scroll->T[Id2+NCV*Itheta]=scroll->T[Ic2[Nc-1]+NCV*Itheta];
    scroll->T[Iddd+NCV*Itheta]=0.0;
    scroll->T[Ic1[Nc-1]+NCV*Itheta]=0.0;
    scroll->T[Ic2[Nc-1]+NCV*Itheta]=0.0;
    scroll->p[Idd+NCV*Itheta]=scroll->p[Iddd+NCV*Itheta];
    scroll->p[Id1+NCV*Itheta]=scroll->p[Ic1[Nc-1]+NCV*Itheta];
    scroll->p[Id2+NCV*Itheta]=scroll->p[Ic2[Nc-1]+NCV*Itheta];
    scroll->p[Iddd+NCV*Itheta]=0.0;
    scroll->p[Ic1[Nc-1]+NCV*Itheta]=0.0;
    scroll->p[Ic2[Nc-1]+NCV*Itheta]=0.0;
    scroll->xL[Idd+NCV*Itheta]=scroll->xL[Iddd+NCV*Itheta];
    scroll->xL[Id1+NCV*Itheta]=scroll->xL[Ic1[Nc-1]+NCV*Itheta];
    scroll->xL[Id2+NCV*Itheta]=scroll->xL[Ic2[Nc-1]+NCV*Itheta];
    scroll->xL[Iddd+NCV*Itheta]=0.0;
    scroll->xL[Ic1[Nc-1]+NCV*Itheta]=0.0;
    scroll->xL[Ic2[Nc-1]+NCV*Itheta]=0.0;
    scroll->m[Idd+NCV*Itheta]=scroll->m[Iddd+NCV*Itheta];
    scroll->m[Id1+NCV*Itheta]=scroll->m[Ic1[Nc-1]+NCV*Itheta];
```

```
    scroll->m[Id2+NCV*Itheta]=scroll->m[Ic2[Nc-1]+NCV*Itheta];
    scroll->m[Iddd+NCV*Itheta]=0.0;
    scroll->m[Ic1[Nc-1]+NCV*Itheta]=0.0;
    scroll->m[Ic2[Nc-1]+NCV*Itheta]=0.0;

    scroll->Forces.Fx[Idd+NCV*Itheta]=scroll->Forces.Fx[Iddd+NCV*Itheta];
    scroll->Forces.Fx[Id1+NCV*Itheta]=scroll->Forces.Fx[Ic1[Nc-1]+NCV*Itheta];
    scroll->Forces.Fx[Id2+NCV*Itheta]=scroll->Forces.Fx[Ic2[Nc-1]+NCV*Itheta];
    scroll->Forces.Fx[Iddd+NCV*Itheta]=0.0;
    scroll->Forces.Fx[Ic1[Nc-1]+NCV*Itheta]=0.0;
    scroll->Forces.Fx[Ic2[Nc-1]+NCV*Itheta]=0.0;

    scroll->Forces.Fy[Idd+NCV*Itheta]=scroll->Forces.Fy[Iddd+NCV*Itheta];
    scroll->Forces.Fy[Id1+NCV*Itheta]=scroll->Forces.Fy[Ic1[Nc-1]+NCV*Itheta];
    scroll->Forces.Fy[Id2+NCV*Itheta]=scroll->Forces.Fy[Ic2[Nc-1]+NCV*Itheta];
    scroll->Forces.Fy[Iddd+NCV*Itheta]=0.0;
    scroll->Forces.Fy[Ic1[Nc-1]+NCV*Itheta]=0.0;
    scroll->Forces.Fy[Ic2[Nc-1]+NCV*Itheta]=0.0;

    scroll->Forces.Fz[Idd+NCV*Itheta]=scroll->Forces.Fz[Iddd+NCV*Itheta];
    scroll->Forces.Fz[Id1+NCV*Itheta]=scroll->Forces.Fz[Ic1[Nc-1]+NCV*Itheta];
    scroll->Forces.Fz[Id2+NCV*Itheta]=scroll->Forces.Fz[Ic2[Nc-1]+NCV*Itheta];
    scroll->Forces.Fz[Iddd+NCV*Itheta]=0.0;
    scroll->Forces.Fz[Ic1[Nc-1]+NCV*Itheta]=0.0;
    scroll->Forces.Fz[Ic2[Nc-1]+NCV*Itheta]=0.0;
}
void Initialize_Masses(struct scrollVals *scroll)
{
    int i;
    double m;
    for(i=0;i<NCV;i++)
    {
        if (scroll->p[i]>0.0)
        {
        m=scroll->V[i]*rho_m(scroll->Ref,scroll->Liq,scroll->T[i],scroll->p[i],scroll->
            xL[i]);
        scroll->m[i]=m;
        }
    }
}
void Initialize_Rotation(struct scrollVals *scroll, struct scrollInputVals *Inputs)
{
    int i;
    double kstar,Vr;
    struct geoVals *geo;

    //Alias pointer to scroll.geo for simplicity
    geo=&(scroll->geo);

    scroll->T[Isa]=Inputs->T_in;
    scroll->p[Isa]=Inputs->p_in;
    scroll->xL[Isa]=Inputs->xL_in;

    scroll->T[Isuction]=Inputs->T_in;
    scroll->p[Isuction]=Inputs->p_in;
    scroll->xL[Isuction]=Inputs->xL_in;

    scroll->T[Is1]=Inputs->T_in;
    scroll->p[Is1]=Inputs->p_in;
    scroll->xL[Is1]=Inputs->xL_in;

    scroll->T[Is2]=Inputs->T_in;
    scroll->p[Is2]=Inputs->p_in;
    scroll->xL[Is2]=Inputs->xL_in;

    if (nC_Max(geo)>0)
    {
        scroll->T[Ic1[0]]=Inputs->T_in;
        scroll->p[Ic1[0]]=Inputs->p_in;
        scroll->xL[Ic1[0]]=Inputs->xL_in;

        scroll->T[Ic2[0]]=Inputs->T_in;
        scroll->p[Ic2[0]]=Inputs->p_in;
        scroll->xL[Ic2[0]]=Inputs->xL_in;

        for (i=1;i<nC(geo,0.0);i++)
        {
            /*
            We want the inlet state for compression chamber #2, so use compression
                chamber #1
            process to determine the state.  Determine the state based on assuming
                constant k
            */
            Vr=Vc(geo,0.0,i)/Vc(geo,2*PI,i);
            kstar=kstar_m(scroll->Ref, scroll->Liq, scroll->T[Ic1[i-1]],scroll->p[Ic1[i
                -1]],scroll->xL[Ic1[i-1]]);
```

```
            scroll->T[Ic1[i]]=scroll->T[Ic1[i-1]]*pow(Vr,kstar-1.0);
            scroll->p[Ic1[i]]=scroll->p[Ic1[i-1]]*pow(Vr,kstar);
            scroll->xL[Ic1[i]]=Inputs->xL_in;
            scroll->T[Ic2[i]]=scroll->T[Ic1[i]];
            scroll->p[Ic2[i]]=scroll->p[Ic1[i]];
            scroll->xL[Ic2[i]]=scroll->xL[Ic1[i]];
        }
        if (theta_d(geo)>7.0/8.0*(2.0*PI) )
        {
            Vr=Vc(geo,0.0,i)/Vc(geo,2*PI,i);
            kstar=kstar_m(scroll->Ref, scroll->Liq, scroll->T[Ic1[i-1]],scroll->p[Ic1[i
                -1]],scroll->xL[Ic1[i-1]]);
            // We are going to start the rotation with all the discharge volumes
            scroll->T[Id1]=scroll->T[Ic1[i-1]]*pow(Vr,kstar-1.0);
            scroll->p[Id1]=scroll->p[Ic1[i-1]]*pow(Vr,kstar);
            scroll->xL[Id1]=Inputs->xL_in;
            scroll->T[Id2]=scroll->T[Id1];
            scroll->p[Id2]=scroll->p[Id1];
            scroll->xL[Id2]=scroll->xL[Id1];
        }
    }
    if (theta_d(geo)<7.0/8.0*(2.0*PI))
    {
        scroll->T[Iddd]=Inputs->T_out;
        scroll->p[Iddd]=Inputs->p_out;
        scroll->xL[Iddd]=Inputs->xL_in;
        scroll->flags.useDDD=true;
    }
    else
    {
        scroll->T[Idd]=Inputs->T_out;
        scroll->p[Idd]=Inputs->p_out;
        scroll->xL[Idd]=Inputs->xL_in;
        scroll->flags.useDDD=false;
    }
    scroll->T[Idischarge]=Inputs->T_out;
    scroll->p[Idischarge]=Inputs->p_out;
    scroll->xL[Idischarge]=Inputs->xL_in;
}
double *Derivs(struct scrollVals * scroll, int Itheta, double theta)
{
    if (ODEVars==T_p_xL)
        return dT_dp_dxL(scroll,Itheta,theta);
    if (ODEVars==T_m_xL)
        return dT_dm_dxL(scroll,Itheta,theta);
}
double *dT_dp_dxL(struct scrollVals * scroll, int Itheta, double theta)
{
     double rho, dudT, u, summerdT, summerdp, summerdxL, drhodT, drhodp, dudp, dudxL,
         drhodxL, *f;
     int i,j;
    double T,p,xL,mdot,h_flow,xL_flow,dTdtheta,dpdtheta,dxLdtheta,V,dV, omega,Q;
    double T_scroll;
    char *Ref, *Liq;
    Ref=scroll->Ref;
    Liq=scroll->Liq;
    dtCounter++;
    /* Run the geometry model to calculate Volumes, derivatives of volumes and leakage
        areas */
    GeometryModel(&(scroll->geo),theta,Itheta,scroll->flags.useDDD,scroll->flags.
        LeftDischarge,scroll->V,scroll->dV,scroll->flowVec);
    /* Run the mass flow model to calculate flows between volumes */
    CalcMassFlows(scroll,Itheta);
    /* Run Scroll-set Heat Transfer model*/
    T_scroll=scroll->HT.T_scroll;
    scrollHT(scroll,theta,Itheta,T_scroll);
    /* Allocate sufficient memory for the derivative vector */
    f=calloc(3*NCV,sizeof(double));
    for (i=0;i<NCV;i++)
    {
        /* If the CV doesn't exist (unused discharge chambers, or suction, discharge, (
            or injection) chamber) */
        if (
            i==Idischarge || i==Isuction
            || (scroll->flags.useDDD==1 && (i==Id1 || i==Id2 || i==Idd))
            || (scroll->flags.useDDD==0 && i==Iddd)
```

```
    || (scroll->flags.LeftDischarge && (i==Ic1[nC_Max(&(scroll->geo))-1] || i==
        Ic2[nC_Max(&(scroll->geo))-1] ) )
    || (theta>theta_d(&(scroll->geo)) && (i==Ic1[nC_Max(&(scroll->geo))-1] || i
        ==Ic2[nC_Max(&(scroll->geo))-1] ) )
    )
{
    /* Put zeros into the derivative vector since the CV doesnt exist */
    f[i]=0.0;
    f[i+NCV]=0.0;
    f[i+2*NCV]=0.0;
}
else
{
    /* Initialize summation terms */
    summerdT=0;
    summerdp=0;
    summerdxL=0;
    /* For each CV, first pull out its T,P,xL,...*/
    T=scroll->T[i+NCV*Itheta];
    p=scroll->p[i+NCV*Itheta];
    xL=scroll->xL[i+NCV*Itheta];
    V=scroll->V[i+NCV*Itheta];
    dV=scroll->dV[i+NCV*Itheta];
    Q=scroll->Q[i+NCV*Itheta];
    /*if (theta>theta_d(&(scroll->geo)))
    {
        printf_plus("I: %d theta: %g// T : %g p: %g xL: %g V: %g dV: %g\n",i,
            theta,T,p,xL,V,dV);
    }*/
    /* Calculate properties and property derivatives
    needed for differential equations */
    rho          =rho_m(Ref,Liq,T,p,xL);
    u         =u_m(Ref,Liq,T,p,xL);
    drhodT       =drhodT_m(Ref,Liq,T,p,xL);
    drhodp       =drhodP_m(Ref,Liq,T,p,xL);
    drhodxL      =drhodxL_m(Ref,Liq,T,p,xL);
    dudT      =dudT_m(Ref,Liq,T,p,xL);
    dudp      =dudP_m(Ref,Liq,T,p,xL);
    dudxL     =dudxL_m(Ref,Liq,T,p,xL);
    omega     =scroll->omega;
    for (j=0;j<scroll->flowVec[Itheta].N;j++)
    {
        mdot=0.0; xL_flow=0.0; h_flow=0.0;
        /* If either CV1 or CV2 is the CV of interest (i), and there is mass flow
            between chambers */
        if ((scroll->flowVec[Itheta].CV1[j]==i || scroll->flowVec[Itheta].CV2[j
            ]==i) && fabs(scroll->flowVec[Itheta].mdot[j])>0)
        {
            /* Pull out the values from the flowVec */
            mdot=scroll->flowVec[Itheta].mdot[j];
            h_flow=scroll->flowVec[Itheta].h_up[j];
            xL_flow=scroll->flowVec[Itheta].xL[j];
            /* If the matching CV is CV2, flip the sign of the mass flow (because
                of sign convention) */
            if (scroll->flowVec[Itheta].CV2[j]==i)
                mdot*=-1;
            //printf_plus("%d,%d,%d:\t%0.12f, \t%0.12f \n",i,scroll->flowVec[
                Itheta].CV1[j],scroll->flowVec[Itheta].CV2[j],mdot,h_flow);
            summerdT+=mdot/omega*( rho*dudp + drhodp*(u-h_flow) + (drhodp*dudxL-
                dudp*drhodxL)*(xL_flow-xL) );
            summerdp+=mdot/omega*( dudT*rho + drhodT*(u-h_flow) + (drhodT*dudxL-
                dudT*drhodxL)*(xL_flow-xL) );
            summerdxL+=mdot/omega*(xL-xL_flow);
        }
    }
    /*if (theta>theta_d(&(scroll->geo))-0.002 && i==Idd)
    {
        printf_plus("I: %d theta: %g// T : %g p: %g xL: %g V: %g dV: %g sT: %g sp
            : %g sxL: %g\n\n",i,theta,T,p,xL,V,dV,summerdT,summerdp,summerdxL);
    }*/
    /* Evaluate the derivatives of the CV */
    dTdtheta=-(-drhodp*Q/omega+drhodp*p*dV-rho*rho*dudp*dV+summerdT)/(rho*V*(
        drhodp*dudT-dudp*drhodT));
    dpdtheta=+(-drhodT*Q/omega+drhodT*p*dV-rho*rho*dudT*dV+summerdp)/(rho*V*(
        drhodp*dudT-dudp*drhodT));
    dxLdtheta=-summerdxL/(rho*V);
```

```
            /* store derivatives */
            f[i]=dTdtheta;
            f[i+NCV]=dpdtheta;
            f[i+2*NCV]=dxLdtheta;
            //printf_plus("CV[%d]\n",i);
            //printf_plus("f_T%g\n",f[i]);
            //printf_plus("f_p%g\n",f[i+NCV]);
            //printf_plus("f_xL%g\n",f[i+2*NCV]);
        }
    }
    return f;
}
double *dT_dm_dxL(struct scrollVals * scroll, int Itheta, double theta)
{
     double rho, dudT, u, h,v,cvm,summerdT, summerdm, summerdxL, dpdT,dudxL, *f;
     int i,j;
    double T,p,xL,m,mdot,h_flow,xL_flow,dTdtheta,dmdtheta,dxLdtheta,V,dV, omega,Q;
    double T_scroll;
    char *Ref, *Liq;
    //if (theta>3.1 && theta<3.8)
    // printf_plus("Into dT_dm_dxL\n");
    Ref=scroll->Ref;
    Liq=scroll->Liq;
    dTmmCounter++;
    /* Run the geometry model to calculate Volumes, derivatives of volumes and leakage
         areas */
    GeometryModel(&(scroll->geo),theta,Itheta,scroll->flags.useDDD,scroll->flags.
        LeftDischarge,scroll->V,scroll->dV,scroll->flowVec);
    /*Calculate the pressures needed */
    for (i=0;i<NCV;i++)
    {
        /* If the suction or discharge, copy the old pressure */
        if (Itheta>0 && (i==Isuction || i==Idischarge) )
            scroll->p[i+NCV*Itheta]=scroll->p[i+NCV*(Itheta-1)];
        /* If the pressure needs to be calculated, calculate it */
        if (Itheta>0 && scroll->T[i+NCV*Itheta]>0 && i!=Isuction && i!=Idischarge)
        {
            rho=scroll->m[i+NCV*Itheta]/scroll->V[i+NCV*Itheta];
            if (scroll->p[i+NCV*Itheta]>1.0)
                scroll->p[i+NCV*Itheta]=p_Trho(scroll->Ref,scroll->Liq,rho,scroll->T[i+
                    NCV*Itheta],scroll->xL[i+NCV*Itheta],scroll->p[i+NCV*Itheta]);
            else
                scroll->p[i+NCV*Itheta]=p_Trho(scroll->Ref,scroll->Liq,rho,scroll->T[i+
                    NCV*Itheta],scroll->xL[i+NCV*Itheta],scroll->p[i+NCV*(Itheta-1)]);
        }
    }
    /* Run the mass flow model to calculate flows between volumes */
    CalcMassFlows(scroll,Itheta);
    /* Run Scroll-set Heat Transfer model*/
    T_scroll=scroll->HT.T_scroll;
    scrollHT(scroll,theta,Itheta,T_scroll);
    /* Allocate sufficient memory for the derivative vector */
    f=(double *)calloc(3*NCV,sizeof(double));
    if (Itheta==0)
        Initialize_Masses(scroll);
    for (i=0;i<NCV;i++)
    {
        /* If the CV doesn't exist (unused discharge chambers, or suction, discharge, (
            or injection) chamber) */
        if (
            i==Idischarge || i==Isuction
            || (scroll->flags.useDDD==1 && (i==Id1 || i==Id2 || i==Idd))
            || (scroll->flags.useDDD==0 && i==Iddd)
            || (scroll->flags.LeftDischarge && (i==Ic1[nC_Max(&(scroll->geo))-1] || i==
                Ic2[nC_Max(&(scroll->geo))-1] ) )
            || (theta>theta_d(&(scroll->geo)) && (i==Ic1[nC_Max(&(scroll->geo))-1] || i
                ==Ic2[nC_Max(&(scroll->geo))-1] ) )
            )
        {
            /* Put zeros into the derivative vector since the CV doesnt exist */
            f[i]=0.0;
            f[i+NCV]=0.0;
            f[i+2*NCV]=0.0;
        }
        else
        {
```

```
            /* Initialize summation terms */
            summerdT=0;
            summerdm=0;
            summerdxL=0;
            /* For each CV, first pull out its T,P,xL,...*/
            T=scroll->T[i+NCV*Itheta];
            p=scroll->p[i+NCV*Itheta];
            xL=scroll->xL[i+NCV*Itheta];
            m=scroll->m[i+NCV*Itheta];
            V=scroll->V[i+NCV*Itheta];
            dV=scroll->dV[i+NCV*Itheta];
            Q=scroll->Q[i+NCV*Itheta];
            /* Calculate properties and property derivatives
            needed for differential equations */
            v         =V/m;
            h         =h_m(Ref,Liq,T,p,xL);
            dpdT      =dpdT_const_v(Ref,Liq,T,p,xL);
            dudxL     =u_m(Ref,Liq,T,p,1.0)-u_m(Ref,Liq,T,p,0.0);
            cvm          =xL*c_l(Liq,T)+(1-xL)*c_v(Ref,T,p);
            omega     =scroll->omega;
            for (j=0;j<scroll->flowVec[Itheta].N;j++)
            {
               mdot=0.0; xL_flow=0.0; h_flow=0.0;
               /* If either CV1 or CV2 is the CV of interest (i), and there is mass flow
                  between chambers */
               if ((scroll->flowVec[Itheta].CV1[j]==i || scroll->flowVec[Itheta].CV2[j
                ]==i) && fabs(scroll->flowVec[Itheta].mdot[j])>0)
               {
                  /* Pull out the values from the flowVec */
                  mdot=scroll->flowVec[Itheta].mdot[j];
                  h_flow=scroll->flowVec[Itheta].h_up[j];
                  xL_flow=scroll->flowVec[Itheta].xL[j];
                  /* If the matching CV is CV2, flip the sign of the mass flow (because
                     of sign convention) */
                  if (scroll->flowVec[Itheta].CV2[j]==i)
                     mdot*=-1;
                  //printf_plus("%d,%d,%d:\t%0.12f, \t%0.12f \n",i,scroll->flowVec[
                      Itheta].CV1[j],scroll->flowVec[Itheta].CV2[j],mdot,h_flow);
                  summerdT += mdot/omega*h_flow;
                  summerdm += mdot/omega;
                  summerdxL += mdot/omega*xL_flow;
               }
            }
            // Evaluate the derivatives of the CV
            dmdtheta=summerdm;
            dxLdtheta=1.0/m*(summerdxL-xL*dmdtheta);
            dTdtheta=1/(m*cvm)*(-T*dpdT*(dV-v*dmdtheta)-m*dudxL*dxLdtheta-h*dmdtheta+Q/
                omega+summerdT);
            // Store derivatives
            f[i]=dTdtheta;
            f[i+NCV]=dmdtheta;
            f[i+2*NCV]=dxLdtheta;
            if (fabs(f[2])>1e6)
            {
               for (j=0;j<scroll->flowVec[Itheta].N;j++)
               {
                  // If either CV1 or CV2 is the CV of interest (i), and there is mass
                      flow between chambers
                  if ((scroll->flowVec[Itheta].CV1[j]==i || scroll->flowVec[Itheta].CV2[
                      j]==i) && fabs(scroll->flowVec[Itheta].mdot[j])>0)
                  {
                  mdot=scroll->flowVec[Itheta].mdot[j];
                  printf_plus("uh oh %g \n",mdot);
                  }
               }
            }
         }
      }
      return f;
}
double calcP(struct scrollVals * scroll, int Itheta)
{
      int i;
      double rho;
      if (ODEVars==T_m_xL)
      {
         for(i=0;i<NCV;i++)
```

```
        {
            if (scroll->T[i+NCV*Itheta]>0)
            {
                rho=scroll->m[i+NCV*Itheta]/scroll->V[i+NCV*Itheta];
                scroll->p[i+NCV*Itheta]=p_Trho(scroll->Ref,scroll->Liq,rho,scroll->T[i+
                    NCV*Itheta],scroll->xL[i+NCV*Itheta],scroll->p[i+NCV*(Itheta-1)]);
            }
        }
    }
}
void freeScroll(struct scrollVals *scroll)
{
    int i;
    free(scroll->T);
    free(scroll->p);
    free(scroll->xL);
    free(scroll->m);
    free(scroll->V);
    free(scroll->dV);
    free(scroll->rho);
    free(scroll->HT.hc);
    free(scroll->HT.A_wall_i);
    free(scroll->HT.A_wall_o);
    free(scroll->HT.Tm_wall_i);
    free(scroll->HT.Tm_wall_o);
    free(scroll->HT.Tm_plate);
    free(scroll->Q);
    free(scroll->error);
    free(scroll->theta);
    free(scroll->Forces.Fx);
    free(scroll->Forces.Fy);
    free(scroll->Forces.Fz);
    free(scroll->Forces.Mx);
    free(scroll->Forces.My);
    free(scroll->Forces.Mz);
    free(scroll->Forces.MO);
    free(scroll->Forces.xcp);
    free(scroll->Forces.ycp);
    for (i=0;i<Ntheta;i++)
    {
        freeFlowVec(&(scroll->flowVec[i]));
    }
    free(scroll->flowVec);
    Ntheta=0;
}
double newTd(struct scrollVals *scrollPtr, double *mdot_out, double *mdot_in)
{
    int i,j;
    double *mdot_disc,*mdoth_disc, *mdot_suct, *mdoth_suct,Tdisc,hdisc;
    struct scrollVals scroll;
    /* Alias copy */
    scroll=*scrollPtr;
    mdoth_disc=calloc(Ntheta,sizeof(double));
    mdot_disc=calloc(Ntheta,sizeof(double));
    mdoth_suct=calloc(Ntheta,sizeof(double));
    mdot_suct=calloc(Ntheta,sizeof(double));
    for (i=0;i<Ntheta;i++)
    {
        for (j=0;j<scroll.flowVec[i].N;j++)
        {
            /* If either of the chambers is the discharge chamber,
            and the other chamber is the dd or ddd chamber */
            if ((scroll.flowVec[i].CV1[j]==Idd || scroll.flowVec[i].CV1[j]==Iddd
                || scroll.flowVec[i].CV2[j]==Idd || scroll.flowVec[i].CV2[j]==Iddd) &&
                (scroll.flowVec[i].CV1[j]==Idischarge || scroll.flowVec[i].CV2[j]==
                    Idischarge))
            {
                mdot_disc[i]=scroll.flowVec[i].mdot[j];
                mdoth_disc[i]=scroll.flowVec[i].mdot[j]*scroll.flowVec[i].h_up[j];
            }
            if ((scroll.flowVec[i].CV1[j]==Isa || scroll.flowVec[i].CV2[j]==Isa) &&
                (scroll.flowVec[i].CV1[j]==Isuction || scroll.flowVec[i].CV2[j]==Isuction
                    ))
            {
                mdot_suct[i]=scroll.flowVec[i].mdot[j];
                mdoth_suct[i]=scroll.flowVec[i].mdot[j]*scroll.flowVec[i].h_up[j];
            }
        }
    }
```

```
    hdisc=trapz(scroll.theta,mdoth_disc,Ntheta)/trapz(scroll.theta,mdot_disc,Ntheta);
    *mdot_out=trapz(scroll.theta,mdot_disc,Ntheta)/(2*PI);
    *mdot_in=trapz(scroll.theta,mdot_suct,Ntheta)/(2*PI);
    Tdisc=T_hp(scroll.Ref,scroll.Liq,hdisc,scroll.p[Idischarge],scroll.xL[Idischarge],
        scroll.T[Idischarge]);
    free(mdot_disc);
    free(mdoth_disc);
    free(mdot_suct);
    free(mdoth_suct);
    return Tdisc;
}
double sumQ(struct scrollVals *scroll)
{
    int i,j; double sum=0.0;
    for (j=0;j<NCV;j++)
    {
        for (i=0;i<Ntheta-1;i++)
        {
            /* Trapezoidal integration for the heat transfer
            for a CV over one rotation */
            sum+=(scroll->Q[j+NCV*(i+1)]+scroll->Q[j+NCV*i])/2.0
                *
                (scroll->theta[i+1]-scroll->theta[i]);
        }
    }
    /*
     * Returned value is positive if the net heat transfer is
     * to the refrigerant/oil mixture
     *
     * The denominator of 2*pi in the return statement
     * gives the average heat transfer over the rotation.
     * The numeric integral of Q gives units of
     * kW-radian.  Dividing by omega [rad/s] gives units of kJ.
     * Dividing by the time of one rotation (2*pi [radians]/omega)
     * yields the result:
     *
     * Q_avg=int(Q*theta)/(2*pi)  in units of [kW]
     *
     */
    return sum / (2*PI);
}
double intPV(double *V, double *p,int N)
{
    int i;
    double sum=0.0;
    // Integrate the volume-pressure data, being careful to only consider parts where
        volume and pressure are defined
    for (i=0;i<N-1;i++)
    {
        if (V[i] > 0.0 && V[i+1] > 0.0 && p[i]>0.0 && p[i+1]>0.0)
            sum+=(p[i]+p[i+1])/2.0*(V[i+1]-V[i]);
    }
    return sum;
}
double f_xcp(struct geoVals geo,double theta, double phi, double phi_0)
{
    double rb,ro,om;
    rb=geo.rb;
    ro=geo.ro;
    om=geo.phi.phi_fie-PI/2.0-theta;
    return (cos(om)*phi*(2*phi_0-phi)*ro-2*(cos(phi)*(phi_0*phi_0-2*phi*phi_0+phi*phi
        -3)+3*sin(phi)*(phi_0-phi))*rb)/(phi*(2*phi_0-phi));
}
double f_ycp(struct geoVals geo,double theta, double phi, double phi_0)
{
    double rb,ro,om;
    rb=geo.rb;
    ro=geo.ro;
    om=geo.phi.phi_fie-PI/2.0-theta;
    return (sin(om)*phi*(2*phi_0-phi)*ro-2*(sin(phi)*(phi_0*phi_0-2*phi*phi_0+phi*phi
        -3)-3*cos(phi)*(phi_0-phi))*rb)/(phi*(2*phi_0-phi));
}
void CalculateForces(struct scrollVals *scroll, int N)
{
    int i,alpha,Nc,LeftDischarge;
    double theta,*col;
    double B,h,ro,rb,phi_e,phi_o0,phi_i0,phi_os,phi_is,phi_ie,b,D,fx_p,fy_p;
    double m_line,ra1,ra2,t1_arc1,t2_arc1,t1_arc2,t2_arc2,t1_line,t2_line;
```

```
double om,x1t,y1t,x2t,y2t,xa1,ya1,xa2,ya2,L,Lx,Ly,nx,ny,x0,y0,Vss1,dVss1;
double cxs1,cys1,cxs2,cys2,Vc1,dVc1,cxc1,cyc1,cxc2,cyc2,Vd1,dVd1,cxd1,cyd1,
   cxd2,cyd2,cxdd,cydd,cxddd,cyddd,rx,ry,sumFxcp,sumFycp,Fx,Fy,xcp,ycp,
   M_O_p,M_O_c1,M_O_c2;
h=scroll->geo.hs;
ro=scroll->geo.ro;
rb=scroll->geo.rb;
phi_e=scroll->geo.phi.phi_fie;
phi_ie=scroll->geo.phi.phi_fie;
phi_o0=scroll->geo.phi.phi_oo0;
phi_i0=scroll->geo.phi.phi_oi0;
phi_os=scroll->geo.phi.phi_oos;
phi_is=scroll->geo.phi.phi_ois;

m_line=scroll->geo.disc.m_line;
xa1=scroll->geo.disc.xa_arc1;
ya1=scroll->geo.disc.ya_arc1;
xa2=scroll->geo.disc.xa_arc2;
ya2=scroll->geo.disc.ya_arc2;

ra1=scroll->geo.disc.ra_arc1;
ra2=scroll->geo.disc.ra_arc2;
t1_arc1=scroll->geo.disc.t1_arc1;
t2_arc1=scroll->geo.disc.t2_arc1;
t1_arc2=scroll->geo.disc.t1_arc2;
t2_arc2=scroll->geo.disc.t2_arc2;

t1_line=scroll->geo.disc.t1_line;
t2_line=scroll->geo.disc.t2_line;

scroll->Forces.sumFx=(double *)calloc(N,sizeof(double));
scroll->Forces.sumFy=(double *)calloc(N,sizeof(double));
scroll->Forces.sumFz=(double *)calloc(N,sizeof(double));
scroll->Forces.sumMx=(double *)calloc(N,sizeof(double));
scroll->Forces.sumMy=(double *)calloc(N,sizeof(double));
scroll->Forces.sumMz=(double *)calloc(N,sizeof(double));
scroll->Forces.sumMO=(double *)calloc(N,sizeof(double));
scroll->Forces.tau=(double *)calloc(N,sizeof(double));
scroll->Forces.Frad=(double *)calloc(N,sizeof(double));

for (i=0;i<N;i++)
{
   theta=scroll->theta[i];
   //Point around which the overturning moments are taken
   x0=scroll->geo.ro*cos(scroll->geo.phi.phi_fie-PI/2.0-theta);
   y0=scroll->geo.ro*sin(scroll->geo.phi.phi_fie-PI/2.0-theta);

   Nc=nC(&(scroll->geo),theta);
   om=phi_e-PI/2-theta;
   // S-SA Break angle calculations
   b=(-phi_o0+phi_e-PI);
   D=ro/rb*((phi_i0-phi_e)*sin(theta)-cos(theta)+1)/(phi_e-phi_i0);
   B=1.0/2.0*(sqrt(b*b-4.0*D)-b);

   // SA chamber
   fx_p=rb*h*(sin(B+phi_e)-(B-phi_o0+phi_e-PI)*cos(B+phi_e)+cos(phi_e)*phi_o0+sin(
       phi_e)-phi_e*cos(phi_e));
   fy_p=-rb*h*(-(-B+phi_o0-phi_e+PI)*sin(B+phi_e)+cos(B+phi_e)-sin(phi_e)*phi_o0+
       phi_e*sin(phi_e)+cos(phi_e));
   M_O_p=-(h*rb*rb*(B-PI)*(B-2*phi_o0+2*phi_e-PI))/2;
   scroll->Forces.Fx[Isa+NCV*i]=fx_p*scroll->p[Isa+NCV*i];
   scroll->Forces.Fy[Isa+NCV*i]=fy_p*scroll->p[Isa+NCV*i];
   scroll->Forces.MO[Isa+NCV*i]=M_O_p*scroll->p[Isa+NCV*i];
   scroll->Forces.Fz[Isa+NCV*i]=scroll->V[Isa+NCV*i]/scroll->geo.hs*scroll->p[Isa+
       NCV*i];
   scroll->Forces.xcp[Isa+NCV*i]=f_xcp(scroll->geo,theta,phi_ie,phi_o0)-f_xcp(
       scroll->geo,theta,phi_ie-PI+B,phi_o0);
   scroll->Forces.ycp[Isa+NCV*i]=f_ycp(scroll->geo,theta,phi_ie,phi_o0)-f_ycp(
       scroll->geo,theta,phi_ie-PI+B,phi_o0);

   // S1 chamber
   fx_p=-rb*h*(sin(B+phi_e)-(B-phi_o0+phi_e-PI)*cos(B+phi_e)+sin(theta-phi_e)-(
       theta+phi_o0-phi_e+PI)*cos(theta-phi_e));
     fy_p=rb*h*((B-phi_o0+phi_e-PI)*sin(B+phi_e)+cos(B+phi_e)-(theta+phi_o0-phi_e+
         PI)*sin(theta-phi_e)-cos(theta-phi_e));
   M_O_p=(h*rb*rb*(B-theta-2*phi_o0+2*phi_e-2*PI)*(B+theta))/2;
     scroll->Forces.Fx[Is1+NCV*i]=fx_p*scroll->p[Is1+NCV*i];
   scroll->Forces.Fy[Is1+NCV*i]=fy_p*scroll->p[Is1+NCV*i];
   scroll->Forces.Fz[Is1+NCV*i]=scroll->V[Is1+NCV*i]/scroll->geo.hs*scroll->p[Is1+
       NCV*i];
   Vs1_calcs(&(scroll->geo),theta,&Vss1,&dVss1,&cxs1,&cys1);
   scroll->Forces.Mx[Is1+NCV*i]= scroll->Forces.Fz[Is1+NCV*i]*(cys1-y0);
   scroll->Forces.My[Is1+NCV*i]=-scroll->Forces.Fz[Is1+NCV*i]*(cxs1-x0);
```

```
scroll->Forces.MO[Is1+NCV*i]=M_O_p*scroll->p[Is1+NCV*i];
scroll->Forces.xcp[Is1+NCV*i]=f_xcp(scroll->geo,theta,phi_ie-PI+B,phi_o0)-f_xcp
    (scroll->geo,theta,phi_ie-PI-theta,phi_o0);
scroll->Forces.ycp[Is1+NCV*i]=f_ycp(scroll->geo,theta,phi_ie-PI+B,phi_o0)-f_ycp
    (scroll->geo,theta,phi_ie-PI-theta,phi_o0);
  // S2 chamber
  fx_p=-rb*h*(sin(theta-phi_e)-(theta+phi_i0-phi_e)*cos(theta-phi_e)+cos(phi_e)
      *(phi_i0-phi_e)+sin(phi_e));
  fy_p=-rb*h*((theta+phi_i0-phi_e)*sin(theta-phi_e)+cos(theta-phi_e)+sin(phi_e)
      *(phi_i0-phi_e)-cos(phi_e));
M_O_p=(h*rb*rb*theta*(theta+2*phi_i0-2*phi_e))/2;
  scroll->Forces.Fx[Is2+NCV*i]=fx_p*scroll->p[Is2+NCV*i];
scroll->Forces.Fy[Is2+NCV*i]=fy_p*scroll->p[Is2+NCV*i];
scroll->Forces.Fz[Is2+NCV*i]=scroll->V[Is2+NCV*i]/scroll->geo.hs*scroll->p[Is2+
    NCV*i];
cxs2=-cxs1+ro*cos(om);
cys2=-cys1+ro*sin(om);
scroll->Forces.Mx[Is2+NCV*i]= scroll->Forces.Fz[Is2+NCV*i]*(cys2-y0);
scroll->Forces.My[Is2+NCV*i]=-scroll->Forces.Fz[Is2+NCV*i]*(cxs2-x0);
scroll->Forces.MO[Is2+NCV*i]=M_O_p*scroll->p[Is2+NCV*i];
scroll->Forces.xcp[Is2+NCV*i]=f_xcp(scroll->geo,theta,phi_ie,phi_i0)-f_xcp(
    scroll->geo,theta,phi_ie-theta,phi_i0);
scroll->Forces.ycp[Is2+NCV*i]=f_ycp(scroll->geo,theta,phi_ie,phi_i0)-f_ycp(
    scroll->geo,theta,phi_ie-theta,phi_i0);
// All of the compression chambers
for (alpha=1;alpha<=Nc;alpha++)
{
   // alpha-1 since C uses 0-based indexing for Ic1 and Ic2
   if (scroll->p[Ic1[alpha-1]+NCV*i]>0.00001)
   {
      fx_p= 2.0*PI*rb*h*cos(theta-phi_e);
      fy_p=-2.0*PI*rb*h*sin(theta-phi_e);
      M_O_c1=-2*PI*h*rb*rb*(theta+phi_o0-phi_e+2*PI*alpha);
      M_O_c2=2*PI*h*rb*rb*(theta+phi_i0-phi_e+2*PI*alpha-PI);
      scroll->Forces.Fx[Ic1[alpha-1]+NCV*i]=fx_p*scroll->p[Ic1[alpha-1]+NCV*i];
      scroll->Forces.Fy[Ic1[alpha-1]+NCV*i]=fy_p*scroll->p[Ic1[alpha-1]+NCV*i];
      scroll->Forces.Fx[Ic2[alpha-1]+NCV*i]=fx_p*scroll->p[Ic2[alpha-1]+NCV*i];
      scroll->Forces.Fy[Ic2[alpha-1]+NCV*i]=fy_p*scroll->p[Ic2[alpha-1]+NCV*i];
      scroll->Forces.MO[Ic1[alpha-1]+NCV*i]=M_O_c1*scroll->p[Ic1[alpha-1]+NCV*i
          ];
      scroll->Forces.MO[Ic2[alpha-1]+NCV*i]=M_O_c2*scroll->p[Ic2[alpha-1]+NCV*i
          ];
      scroll->Forces.Fz[Ic1[alpha-1]+NCV*i]=scroll->V[Ic1[alpha-1]+NCV*i]/
          scroll->geo.hs*scroll->p[Ic1[alpha-1]+NCV*i];
      scroll->Forces.Fz[Ic2[alpha-1]+NCV*i]=scroll->V[Ic2[alpha-1]+NCV*i]/
          scroll->geo.hs*scroll->p[Ic2[alpha-1]+NCV*i];
      scroll->Forces.xcp[Ic1[alpha-1]+NCV*i]=f_xcp(scroll->geo,theta,phi_ie-
          theta-2*PI*alpha-PI,phi_o0)-f_xcp(scroll->geo,theta,phi_ie-theta-2*PI
          *(alpha-1)-PI,phi_o0);
      scroll->Forces.ycp[Ic1[alpha-1]+NCV*i]=f_ycp(scroll->geo,theta,phi_ie-
          theta-2*PI*alpha-PI,phi_o0)-f_ycp(scroll->geo,theta,phi_ie-theta-2*PI
          *(alpha-1)-PI,phi_o0);
      scroll->Forces.xcp[Ic2[alpha-1]+NCV*i]=f_xcp(scroll->geo,theta,phi_ie-
          theta-2*PI*alpha,phi_i0)-f_xcp(scroll->geo,theta,phi_ie-theta-2*PI*(
          alpha-1),phi_i0);
      scroll->Forces.ycp[Ic2[alpha-1]+NCV*i]=f_ycp(scroll->geo,theta,phi_ie-
          theta-2*PI*alpha,phi_i0)-f_ycp(scroll->geo,theta,phi_ie-theta-2*PI*(
          alpha-1),phi_i0);
      Vc1_calcs(&(scroll->geo),theta,alpha,&Vc1,&dVc1,&cxc1,&cyc1);
      cxc2=-cxc1+ro*cos(om);
      cyc2=-cyc1+ro*sin(om);
      scroll->Forces.Mx[Ic1[alpha-1]+NCV*i]= scroll->Forces.Fz[Ic1[alpha-1]+NCV
          *i]*(cyc1-y0);
      scroll->Forces.My[Ic1[alpha-1]+NCV*i]=-scroll->Forces.Fz[Ic1[alpha-1]+NCV
          *i]*(cxc1-x0);
      scroll->Forces.Mx[Ic2[alpha-1]+NCV*i]= scroll->Forces.Fz[Ic2[alpha-1]+NCV
          *i]*(cyc2-y0);
      scroll->Forces.My[Ic2[alpha-1]+NCV*i]=-scroll->Forces.Fz[Ic2[alpha-1]+NCV
          *i]*(cxc2-x0);
   }
   else
   {
      scroll->Forces.Fx[Ic1[alpha-1]+NCV*i]=0.0;
      scroll->Forces.Fy[Ic1[alpha-1]+NCV*i]=0.0;
      scroll->Forces.Fx[Ic2[alpha-1]+NCV*i]=0.0;
```

```
      scroll->Forces.Fy[Ic2[alpha-1]+NCV*i]=0.0;
      scroll->Forces.Fz[Ic1[alpha-1]+NCV*i]=0.0;
      scroll->Forces.Fz[Ic2[alpha-1]+NCV*i]=0.0;
      scroll->Forces.Mx[Ic1[alpha-1]+NCV*i]=0.0;
      scroll->Forces.My[Ic1[alpha-1]+NCV*i]=0.0;
      scroll->Forces.Mx[Ic2[alpha-1]+NCV*i]=0.0;
      scroll->Forces.My[Ic2[alpha-1]+NCV*i]=0.0;
      scroll->Forces.MO[Ic1[alpha-1]+NCV*i]=0.0;
      scroll->Forces.MO[Ic2[alpha-1]+NCV*i]=0.0;
      scroll->Forces.Mz[Ic1[alpha-1]+NCV*i]=0.0;
      scroll->Forces.Mz[Ic2[alpha-1]+NCV*i]=0.0;
   }
}
sumFxcp=0.0;
sumFycp=0.0;
// Check if DDD chamber is being used at this step
if (scroll->p[Id1+NCV*i]>0.00001)
{
   //  DDD not being used, D1 and D2 and DD are all separate
   // Check if theta brackets the discharge angle
   if (scroll->theta[i]<theta_d(&(scroll->geo)) && scroll->theta[i+1]>theta_d
       (&(scroll->geo)))
      LeftDischarge=true;
   else
      LeftDischarge=false;
   // D1 chamber
   if (LeftDischarge==true)
   {
      fx_p= 2.0*PI*rb*h*cos(theta-phi_e);
      fy_p=-2.0*PI*rb*h*sin(theta-phi_e);
      M_O_p=-2*PI*h*rb*rb*(theta+phi_o0-phi_e+2*PI*Nc);
   }
   else
   {
      fx_p= rb*h*(sin(theta-phi_e)+(-theta-phi_o0+phi_e-2*PI*Nc-PI)*cos(theta-
          phi_e)-sin(phi_os)-(phi_o0-phi_os)*cos(phi_os));
      fy_p=-rb*h*((-theta-phi_o0+phi_e-2*PI*Nc-PI)*sin(theta-phi_e)-cos(theta-
          phi_e)-(phi_os-phi_o0)*sin(phi_os)-cos(phi_os));
      M_O_p=(h*rb*rb*(theta-phi_os+2*phi_o0-phi_e+2*PI*Nc+PI)*(theta+phi_os-
          phi_e+2*PI*Nc+PI))/2.0;
   }
   scroll->Forces.Fx[Id1+NCV*i]=fx_p*scroll->p[Id1+NCV*i];
   scroll->Forces.Fy[Id1+NCV*i]=fy_p*scroll->p[Id1+NCV*i];
   scroll->Forces.MO[Id1+NCV*i]=M_O_p*scroll->p[Id1+NCV*i];
   scroll->Forces.Fz[Id1+NCV*i]=scroll->V[Id1+NCV*i]/scroll->geo.hs*scroll->p[
       Id1+NCV*i];
   Vd1_calcs(&(scroll->geo),theta,&Vd1,&dVd1,&cxd1,&cyd1);
   scroll->Forces.Mx[Id1+NCV*i]= scroll->Forces.Fz[Id1+NCV*i]*(cyd1-y0);
   scroll->Forces.My[Id1+NCV*i]=-scroll->Forces.Fz[Id1+NCV*i]*(cxd1-x0);
   scroll->Forces.xcp[Id1+NCV*i]=f_xcp(scroll->geo,theta,phi_ie-theta-2*PI*Nc-
       PI,phi_o0)-f_xcp(scroll->geo,theta,phi_os,phi_o0);
   scroll->Forces.ycp[Id1+NCV*i]=f_ycp(scroll->geo,theta,phi_ie-theta-2*PI*Nc-
       PI,phi_o0)-f_ycp(scroll->geo,theta,phi_os,phi_o0);
   // D2 chamber
   if (LeftDischarge==true)
   {
      fx_p= 2.0*PI*rb*h*cos(theta-phi_e);
      fy_p=-2.0*PI*rb*h*sin(theta-phi_e);
      M_O_p=2*PI*h*rb*rb*(theta+phi_i0-phi_e+2*PI*Nc-PI);
   }
   else
   {
      fx_p=-rb*h*(-sin(theta-phi_e)+(theta+phi_i0-phi_e+2*PI*Nc)*cos(theta-
          phi_e)+sin(phi_os)-(phi_os-phi_i0+PI)*cos(phi_os));
      fy_p=rb*h*((theta+phi_i0-phi_e+2*PI*Nc)*sin(theta-phi_e)+cos(theta-phi_e)
          -(-phi_os+phi_i0-PI)*sin(phi_os)+cos(phi_os));
      M_O_p=-(h*rb*rb*(theta-phi_os+2*phi_i0-phi_e+2*PI*Nc-PI)*(theta+phi_os-
          phi_e+2*PI*Nc+PI))/2;
   }
   scroll->Forces.Fx[Id2+NCV*i]=fx_p*scroll->p[Id2+NCV*i];
   scroll->Forces.Fy[Id2+NCV*i]=fy_p*scroll->p[Id2+NCV*i];
   scroll->Forces.MO[Id2+NCV*i]=M_O_p*scroll->p[Id2+NCV*i];
   scroll->Forces.Fz[Id2+NCV*i]=scroll->V[Id2+NCV*i]/scroll->geo.hs*scroll->p[
       Id2+NCV*i];
   cxd2=-cxd1+ro*cos(om);
   cyd2=-cyd1+ro*sin(om);
   scroll->Forces.Mx[Id2+NCV*i]= scroll->Forces.Fz[Id2+NCV*i]*(cyd2-y0);
```

```
scroll->Forces.My[Id2+NCV*i]=-scroll->Forces.Fz[Id2+NCV*i]*(cxd2-x0);
scroll->Forces.xcp[Id2+NCV*i]=f_xcp(scroll->geo,theta,phi_ie-theta-2*PI*Nc,
    phi_i0)-f_xcp(scroll->geo,theta,phi_os+PI,phi_i0);
scroll->Forces.ycp[Id2+NCV*i]=f_ycp(scroll->geo,theta,phi_ie-theta-2*PI*Nc,
    phi_i0)-f_ycp(scroll->geo,theta,phi_os+PI,phi_i0);
// DD Chamber - Involute small segment
fx_p=-rb*h*(-sin(phi_os)+(phi_os-phi_i0+PI)*cos(phi_os)-sin(phi_is)-(phi_i0-
    phi_is)*cos(phi_is));
   fy_p= rb*h*((-phi_os+phi_i0-PI)*sin(phi_os)-cos(phi_os)-(phi_is-phi_i0)*
       sin(phi_is)-cos(phi_is));
M_O_p=-(h*rb*rb*(phi_os-phi_is+PI)*(phi_os+phi_is-2*phi_i0+PI))/2;
Fx=fx_p*scroll->p[Idd+NCV*i];
Fy=fy_p*scroll->p[Idd+NCV*i];
scroll->Forces.Fx[Idd+NCV*i]=Fx;
scroll->Forces.Fy[Idd+NCV*i]=Fy;
scroll->Forces.MO[Idd+NCV*i]=M_O_p*scroll->p[Idd+NCV*i];
scroll->Forces.Fz[Idd+NCV*i]=scroll->V[Idd+NCV*i]/scroll->geo.hs*scroll->p[
    Idd+NCV*i];
cxdd=ro/2*cos(om);
cydd=ro/2*sin(om);
scroll->Forces.Mx[Idd+NCV*i]= scroll->Forces.Fz[Idd+NCV*i]*(cydd-y0);
scroll->Forces.My[Idd+NCV*i]=-scroll->Forces.Fz[Idd+NCV*i]*(cxdd-x0);
sumFxcp=sqrt(Fx*Fx+Fy*Fy)*(f_xcp(scroll->geo,theta,phi_os+PI,phi_i0)-f_xcp(
    scroll->geo,theta,phi_is,phi_i0));
sumFycp=sqrt(Fx*Fx+Fy*Fy)*(f_ycp(scroll->geo,theta,phi_os+PI,phi_i0)-f_ycp(
    scroll->geo,theta,phi_is,phi_i0));
   // DD chamber - Arc 1
   fx_p=-h*ra1*(sin(t2_arc1)-sin(t1_arc1));
   fy_p=+h*ra1*(cos(t2_arc1)-cos(t1_arc1));
M_O_p=-h*ra1*((sin(t2_arc1)-sin(t1_arc1))*ya1+(cos(t2_arc1)-cos(t1_arc1))*
    xa1);
Fx=fx_p*scroll->p[Idd+NCV*i];
Fy=fy_p*scroll->p[Idd+NCV*i];
   scroll->Forces.Fx[Idd+NCV*i]=scroll->Forces.Fx[Idd+NCV*i]+Fx;
scroll->Forces.Fy[Idd+NCV*i]=scroll->Forces.Fy[Idd+NCV*i]+Fy;
scroll->Forces.MO[Idd+NCV*i]=scroll->Forces.MO[Idd+NCV*i]+M_O_p*scroll->p[
    Idd+NCV*i];
xcp=-xa1+ro*cos(om)-ra1*(sin(t2_arc1)-sin(t1_arc1))/(t2_arc1-t1_arc1);
ycp=-ya1+ro*sin(om)+ra1*(cos(t2_arc1)-cos(t1_arc1))/(t2_arc1-t1_arc1);
sumFxcp=sumFxcp+sqrt(Fx*Fx+Fy*Fy)*xcp;
sumFycp=sumFxcp+sqrt(Fx*Fx+Fy*Fy)*ycp;
   // DD chamber - Arc 2
   fx_p=+h*ra2*(sin(t2_arc2)-sin(t1_arc2));
   fy_p=-h*ra2*(cos(t2_arc2)-cos(t1_arc2));
M_O_p=+h*ra2*((sin(t2_arc2)-sin(t1_arc2))*ya2+(cos(t2_arc2)-cos(t1_arc2))*
    xa2);
Fx=fx_p*scroll->p[Idd+NCV*i];
Fy=fy_p*scroll->p[Idd+NCV*i];
scroll->Forces.Fx[Idd+NCV*i]=scroll->Forces.Fx[Idd+NCV*i]+Fx;
scroll->Forces.Fy[Idd+NCV*i]=scroll->Forces.Fy[Idd+NCV*i]+Fy;
scroll->Forces.MO[Idd+NCV*i]=scroll->Forces.MO[Idd+NCV*i]+M_O_p*scroll->p[
    Idd+NCV*i];
xcp=-xa2+ro*cos(om)-ra2*(sin(t2_arc2)-sin(t1_arc2))/(t2_arc2-t1_arc2);
ycp=-ya2+ro*sin(om)+ra2*(cos(t2_arc2)-cos(t1_arc2))/(t2_arc2-t1_arc2);
sumFxcp=sumFxcp+sqrt(Fx*Fx+Fy*Fy)*xcp;
sumFycp=sumFxcp+sqrt(Fx*Fx+Fy*Fy)*ycp;
   // DD chamber - Line
x1t=-xa1-ra1*cos(t1_arc1)+ro*cos(om);
y1t=-ya1-ra1*sin(t1_arc1)+ro*sin(om);
x2t=-xa2-ra2*cos(t1_arc2)+ro*cos(om);
y2t=-ya2-ra2*sin(t1_arc2)+ro*sin(om);
L=sqrt(pow(x2t-x1t,2)+pow(y2t-y1t,2));
// If the length of the line is zero, then the normal force
// components are undefined, so don't let it calculate them
if (L>1e-12)
{
   Lx=(x2t-x1t)/L;
   Ly=(y2t-y1t)/L;
   nx=-1/sqrt(1+pow(Lx,2)/pow(Ly,2));
   ny=Lx/Ly/sqrt(1+pow(Lx,2)/pow(Ly,2));
   // Make sure you get the cross product with the normal
   // pointing towards the scroll, otherwise flip...
   if (Lx*ny-Ly*nx<0)
   {  nx=-nx; ny=-ny; }
   fx_p=h*nx*L;
```

```
            fy_p=h*ny*L;
        }
        else
        {
            fx_p=0.0;
            fy_p=0.0;
        }
        M_O_p=((x1t+x2t)/2.0-ro*cos(om))*fy_p-((y1t+y2t)/2.0-ro*sin(om))*fx_p;
        Fx=fx_p*scroll->p[Idd+NCV*i];
        Fy=fy_p*scroll->p[Idd+NCV*i];
            scroll->Forces.Fx[Idd+NCV*i]=scroll->Forces.Fx[Idd+NCV*i]+Fx;
        scroll->Forces.Fy[Idd+NCV*i]=scroll->Forces.Fy[Idd+NCV*i]+Fy;
        scroll->Forces.MO[Idd+NCV*i]=scroll->Forces.MO[Idd+NCV*i]+M_O_p*scroll->p[
            Idd+NCV*i];
        sumFxcp=sumFxcp+sqrt(Fx*Fx+Fy*Fy)*(x1t+x2t)/2.0;
        sumFycp=sumFxcp+sqrt(Fx*Fx+Fy*Fy)*(y1t+y2t)/2.0;
        // Calculate the center of pressure from the gas forces on the discharge
            region
        Fx=scroll->Forces.Fx[Idd+NCV*i];
        Fy=scroll->Forces.Fy[Idd+NCV*i];
        scroll->Forces.xcp[Idd+NCV*i]=sumFxcp/sqrt(Fx*Fx+Fy*Fy);
        scroll->Forces.ycp[Idd+NCV*i]=sumFycp/sqrt(Fx*Fx+Fy*Fy);
        scroll->Forces.Fx[Iddd+NCV*i]=0.0;
        scroll->Forces.Fy[Iddd+NCV*i]=0.0;
    }
    else
    {
        scroll->Forces.Fz[Iddd+NCV*i]=scroll->V[Iddd+NCV*i]/scroll->geo.hs*scroll->p
            [Iddd+NCV*i];
        cxddd=ro/2*cos(om);
        cyddd=ro/2*sin(om);
        scroll->Forces.Mx[Iddd+NCV*i]= scroll->Forces.Fz[Iddd+NCV*i]*(cyddd-y0);
        scroll->Forces.My[Iddd+NCV*i]=-scroll->Forces.Fz[Iddd+NCV*i]*(cxddd-x0);
        // DDD chamber - D1 part
            fx_p= rb*h*(sin(theta-phi_e)+(-theta-phi_o0+phi_e-2*PI*Nc-PI)*cos(theta-
                phi_e)-sin(phi_os)-(phi_o0-phi_os)*cos(phi_os));
            fy_p=-rb*h*((-theta-phi_o0+phi_e-2*PI*Nc-PI)*sin(theta-phi_e)-cos(theta-
                phi_e)-(phi_os-phi_o0)*sin(phi_os)-cos(phi_os));
        M_O_p=(h*rb*rb*(theta-phi_os+2*phi_o0-phi_e+2*PI*Nc+PI)*(theta+phi_os-phi_e
            +2*PI*Nc+PI))/2.0;
        scroll->Forces.Fx[Iddd+NCV*i]=fx_p*scroll->p[Iddd+NCV*i];
        scroll->Forces.Fy[Iddd+NCV*i]=fy_p*scroll->p[Iddd+NCV*i];
        scroll->Forces.MO[Iddd+NCV*i]=M_O_p*scroll->p[Iddd+NCV*i];
        Fx=fx_p*scroll->p[Iddd+NCV*i];
        Fy=fx_p*scroll->p[Iddd+NCV*i];
        xcp=f_xcp(scroll->geo,theta,phi_ie-theta-2*PI*Nc-PI,phi_o0)-f_xcp(scroll->
            geo,theta,phi_os,phi_o0);
        ycp=f_ycp(scroll->geo,theta,phi_ie-theta-2*PI*Nc-PI,phi_o0)-f_ycp(scroll->
            geo,theta,phi_os,phi_o0);
        sumFxcp=sumFxcp+sqrt(Fx*Fx+Fy*Fy)*xcp;
        sumFycp=sumFxcp+sqrt(Fx*Fx+Fy*Fy)*ycp;
        // DDD chamber - D2 part
            fx_p=-rb*h*(-sin(theta-phi_e)+(theta+phi_i0-phi_e+2*PI*Nc)*cos(theta-
                phi_e)+sin(phi_os)-(phi_os-phi_i0+PI)*cos(phi_os));
            fy_p=rb*h*((theta+phi_i0-phi_e+2*PI*Nc)*sin(theta-phi_e)+cos(theta-phi_e)
                -(-phi_os+phi_i0-PI)*sin(phi_os)+cos(phi_os));
        M_O_p=-(h*rb*rb*(theta-phi_os+2*phi_i0-phi_e+2*PI*Nc-PI)*(theta+phi_os-phi_e
            +2*PI*Nc+PI))/2;
        scroll->Forces.Fx[Iddd+NCV*i]=scroll->Forces.Fx[Iddd+NCV*i]+fx_p*scroll->p[
            Iddd+NCV*i];
        scroll->Forces.Fy[Iddd+NCV*i]=scroll->Forces.Fy[Iddd+NCV*i]+fy_p*scroll->p[
            Iddd+NCV*i];
        scroll->Forces.MO[Iddd+NCV*i]=scroll->Forces.MO[Iddd+NCV*i]+M_O_p*scroll->p[
            Iddd+NCV*i];
        Fx=fx_p*scroll->p[Iddd+NCV*i];
        Fy=fx_p*scroll->p[Iddd+NCV*i];
        xcp=f_xcp(scroll->geo,theta,phi_ie-theta-2*PI*Nc,phi_i0)-f_xcp(scroll->geo,
            theta,phi_os+PI,phi_i0);
        ycp=f_ycp(scroll->geo,theta,phi_ie-theta-2*PI*Nc,phi_i0)-f_ycp(scroll->geo,
            theta,phi_os+PI,phi_i0);
        sumFxcp=sumFxcp+sqrt(Fx*Fx+Fy*Fy)*xcp;
        sumFycp=sumFxcp+sqrt(Fx*Fx+Fy*Fy)*ycp;
        // DDD Chamber - Involute small segment
        fx_p=-rb*h*(-sin(phi_os)+(phi_os-phi_i0+PI)*cos(phi_os)-sin(phi_is)-(phi_i0-
            phi_is)*cos(phi_is));
```

```
   fy_p= rb*h*((-phi_os+phi_i0-PI)*sin(phi_os)-cos(phi_os)-(phi_is-phi_i0)*
       sin(phi_is)-cos(phi_is));
M_O_p=-(h*(phi_os-phi_is+PI)*(phi_os+phi_is-2*phi_i0+PI)*rb*rb)/2;
scroll->Forces.Fx[Iddd+NCV*i]=scroll->Forces.Fx[Iddd+NCV*i]+fx_p*scroll->p[
    Iddd+NCV*i];
scroll->Forces.Fy[Iddd+NCV*i]=scroll->Forces.Fy[Iddd+NCV*i]+fy_p*scroll->p[
    Iddd+NCV*i];
scroll->Forces.MO[Iddd+NCV*i]=scroll->Forces.MO[Iddd+NCV*i]+M_O_p*scroll->p[
    Iddd+NCV*i];
Fx=fx_p*scroll->p[Iddd+NCV*i];
Fy=fx_p*scroll->p[Iddd+NCV*i];
xcp=f_xcp(scroll->geo,theta,phi_os+PI,phi_i0)-f_xcp(scroll->geo,theta,phi_is
    ,phi_i0);
ycp=f_ycp(scroll->geo,theta,phi_os+PI,phi_i0)-f_ycp(scroll->geo,theta,phi_is
    ,phi_i0);
sumFxcp=sumFxcp+sqrt(Fx*Fx+Fy*Fy)*xcp;
sumFycp=sumFxcp+sqrt(Fx*Fx+Fy*Fy)*ycp;
   // DDD chamber - Arc 1
   fx_p=-h*ra1*(sin(t2_arc1)-sin(t1_arc1));
   fy_p=+h*ra1*(cos(t2_arc1)-cos(t1_arc1));
M_O_p=-h*ra1*((sin(t2_arc1)-sin(t1_arc1))*ya1+(cos(t2_arc1)-cos(t1_arc1))*
    xa1);
   scroll->Forces.Fx[Iddd+NCV*i]=scroll->Forces.Fx[Iddd+NCV*i]+fx_p*scroll->
       p[Iddd+NCV*i];
scroll->Forces.Fy[Iddd+NCV*i]=scroll->Forces.Fy[Iddd+NCV*i]+fy_p*scroll->p[
    Iddd+NCV*i];
scroll->Forces.MO[Iddd+NCV*i]=scroll->Forces.MO[Iddd+NCV*i]+M_O_p*scroll->p[
    Iddd+NCV*i];
Fx=fx_p*scroll->p[Iddd+NCV*i];
Fy=fx_p*scroll->p[Iddd+NCV*i];
   xcp=-xa1+ro*cos(om)-ra1*(sin(t2_arc1)-sin(t1_arc1))/(t2_arc1-t1_arc1);
ycp=-ya1+ro*sin(om)+ra1*(cos(t2_arc1)-cos(t1_arc1))/(t2_arc1-t1_arc1);
sumFxcp=sumFxcp+sqrt(Fx*Fx+Fy*Fy)*xcp;
sumFycp=sumFxcp+sqrt(Fx*Fx+Fy*Fy)*ycp;
   // DDD chamber - Arc 2
   fx_p=+h*ra2*(sin(t2_arc2)-sin(t1_arc2));
   fy_p=-h*ra2*(cos(t2_arc2)-sin(t1_arc2));
M_O_p=+h*ra2*((sin(t2_arc2)-sin(t1_arc2))*ya2+(cos(t2_arc2)-cos(t1_arc2))*
    xa2);
scroll->Forces.Fx[Iddd+NCV*i]=scroll->Forces.Fx[Iddd+NCV*i]+fx_p*scroll->p[
    Iddd+NCV*i];
scroll->Forces.Fy[Iddd+NCV*i]=scroll->Forces.Fy[Iddd+NCV*i]+fy_p*scroll->p[
    Iddd+NCV*i];
scroll->Forces.MO[Iddd+NCV*i]=scroll->Forces.MO[Iddd+NCV*i]+M_O_p*scroll->p[
    Iddd+NCV*i];
Fx=fx_p*scroll->p[Iddd+NCV*i];
Fy=fx_p*scroll->p[Iddd+NCV*i];
   xcp=-xa2+ro*cos(om)-ra2*(sin(t2_arc2)-sin(t1_arc2))/(t2_arc2-t1_arc2);
ycp=-ya2+ro*sin(om)+ra2*(cos(t2_arc2)-cos(t1_arc2))/(t2_arc2-t1_arc2);
sumFxcp=sumFxcp+sqrt(Fx*Fx+Fy*Fy)*xcp;
sumFycp=sumFxcp+sqrt(Fx*Fx+Fy*Fy)*ycp;
   // DDD chamber - Line
x1t=-xa1-ra1*cos(t1_arc1)+ro*cos(om);
y1t=-ya1-ra1*sin(t1_arc1)+ro*sin(om);
x2t=-xa2-ra2*cos(t1_arc2)+ro*cos(om);
y2t=-ya2-ra2*sin(t1_arc2)+ro*sin(om);
L=sqrt(pow(x2t-x1t,2)+pow(y2t-y1t,2));
// If the length of the line is zero, then the normal force
// components are undefined, so don't let it calculate them
if (L>1e-12)
{
   Lx=(x2t-x1t)/L;
   Ly=(y2t-y1t)/L;
   // Make sure it isn't a vertical line with y1t=y2t
   if (fabs(Ly)>1e-12)
   {
      // Not a vertical line
      nx=-1/sqrt(1+pow(Lx,2)/pow(Ly,2));
      ny=Lx/Ly/sqrt(1+pow(Lx,2)/pow(Ly,2));
   }
   else
   {
      // A vertical line
      // Code below will take care of the sign on normal
      nx=1;
      ny=0;
```

```
            }
            // Make sure you get the cross product with the normal
            // pointing towards the scroll, otherwise flip...
            if (Lx*ny-Ly*nx<0)
            {  nx=-nx; ny=-ny; }
            fx_p=h*nx*L;
            fy_p=h*ny*L;
         }
         else
         {
            fx_p=0.0;
            fy_p=0.0;
         }
         M_O_p=((x1t+x2t)/2.0-ro*cos(om))*fy_p-((y1t+y2t)/2.0-ro*sin(om))*fx_p;
         Fx=fx_p*scroll->p[Iddd+NCV*i];
         Fy=fy_p*scroll->p[Iddd+NCV*i];
            scroll->Forces.Fx[Iddd+NCV*i]=scroll->Forces.Fx[Iddd+NCV*i]+Fx;
         scroll->Forces.Fy[Iddd+NCV*i]=scroll->Forces.Fy[Iddd+NCV*i]+Fy;
         scroll->Forces.MO[Iddd+NCV*i]=scroll->Forces.MO[Iddd+NCV*i]+M_O_p*scroll->p[
             Iddd+NCV*i];
         sumFxcp=sumFxcp+sqrt(Fx*Fx+Fy*Fy)*(x1t+x2t)/2.0;
         sumFycp=sumFxcp+sqrt(Fx*Fx+Fy*Fy)*(y1t+y2t)/2.0;
         // Calculate the center of pressure from the gas forces on the discharge
             region
         Fx=scroll->Forces.Fx[Iddd+NCV*i];
         Fy=scroll->Forces.Fy[Iddd+NCV*i];
         scroll->Forces.xcp[Iddd+NCV*i]=sumFxcp/sqrt(Fx*Fx+Fy*Fy);
         scroll->Forces.ycp[Iddd+NCV*i]=sumFycp/sqrt(Fx*Fx+Fy*Fy);
         scroll->Forces.Fx[Id1+NCV*i]=0.0;
         scroll->Forces.Fy[Id1+NCV*i]=0.0;
         scroll->Forces.Fx[Id2+NCV*i]=0.0;
         scroll->Forces.Fy[Id2+NCV*i]=0.0;
         scroll->Forces.Fx[Idd+NCV*i]=0.0;
         scroll->Forces.Fy[Idd+NCV*i]=0.0;
      }
      // Sum up the forces in the x direction
      col=colSlice(scroll->Forces.Fx,NCV,N,i);
      scroll->Forces.sumFx[i]=sumVector(col,NCV);
      free(col);
      // Sum up the forces in the y direction
      col=colSlice(scroll->Forces.Fy,NCV,N,i);
      scroll->Forces.sumFy[i]=sumVector(col,NCV);
      free(col);
      // Sum up the forces in the z direction
      col=colSlice(scroll->Forces.Fz,NCV,N,i);
      scroll->Forces.sumFz[i]=sumVector(col,NCV);
      free(col);
      // Sum up the forces in the z direction
      col=colSlice(scroll->Forces.MO,NCV,N,i);
      scroll->Forces.sumMO[i]=sumVector(col,NCV);
      free(col);
      // Calculate the instantaneous torque about the origin
      rx=ro*cos(om);
      ry=ro*sin(om);
      scroll->Forces.tau[i]=rx*(scroll->Forces.sumFy[i])-ry*(scroll->Forces.sumFx[i])
          ;
      scroll->Forces.Frad[i]=sqrt(pow(scroll->Forces.sumFy[i],2)+pow(scroll->Forces.
          sumFx[i],2));
   }
   //Calculate average values over the course of one rotation
   scroll->Forces.Fx_mean=trapz(scroll->theta,scroll->Forces.sumFx,N)/(2*PI);
   scroll->Forces.Fy_mean=trapz(scroll->theta,scroll->Forces.sumFy,N)/(2*PI);
   scroll->Forces.Fz_mean=trapz(scroll->theta,scroll->Forces.sumFz,N)/(2*PI);
   scroll->Forces.Frad_mean=trapz(scroll->theta,scroll->Forces.Frad,N)/(2*PI);
   scroll->Forces.MO_mean=trapz(scroll->theta,scroll->Forces.sumMO,N)/(2*PI);
   scroll->Forces.tau_mean=trapz(scroll->theta,scroll->Forces.tau,N)/(2*PI);
   // Free the allocated vectors
   free(scroll->Forces.sumFx);
   free(scroll->Forces.sumFy);
   free(scroll->Forces.sumFz);
   free(scroll->Forces.sumMx);
   free(scroll->Forces.sumMy);
   free(scroll->Forces.sumMz);
   free(scroll->Forces.sumMO);
   free(scroll->Forces.tau);
```

```
    free(scroll->Forces.Frad);
}
void CalculateLossTerms(struct scrollVals *scroll)
{
    int i,N,j,iD;
    double *xRadial,*yRadial,*xFlank,*yFlank,*xSuction, *ySuction,
        e1,e2,h1,h2s,s1,V1,Vcl,T1,xL1,P1,P2,V2,PV_ideal,k,*xd1,*yd1,
        *xd2,*yd2,*xdd,*ydd,*xddd,*yddd,PV_actual;
    // ****************************
    // Calculation of adiabatic power
    // ****************************
    s1=s_m(scroll->Ref,scroll->Liq,scroll->T[Isuction],scroll->p[Isuction],scroll->xL[
        Isuction]);
    h1=h_m(scroll->Ref,scroll->Liq,scroll->T[Isuction],scroll->p[Isuction],scroll->xL[
        Isuction]);
    h2s=h_sp(scroll->Ref,scroll->Liq,s1,scroll->p[Idischarge],scroll->xL[Idischarge],
        scroll->T[Idischarge]);
    scroll->Losses.Wdot_adiabatic=scroll->massFlow.mdot_tot*(h2s-h1);
    // ****************************
    // Calculation of mechanical losses
    // ****************************
    // Already calculated, but copy it over anyway
    scroll->Losses.mechanical=scroll->PowerEff.P_ML;
    // ****************************
    // Calculation of leakage losses
    // ****************************
    //Allocate empty temporary vectors
    xRadial=(double *)malloc(scroll->Ntheta*sizeof(double));
    yRadial=(double *)malloc(scroll->Ntheta*sizeof(double));
    xFlank=(double *)malloc(scroll->Ntheta*sizeof(double));
    yFlank=(double *)malloc(scroll->Ntheta*sizeof(double));
    // Loop over all points of the revolution
    for (i=0;i<scroll->Ntheta;i++)
    {
        // Initialize terms (malloc doesn't seem to zero out the y vectors)
        xRadial[i]=0.0; yRadial[i]=0.0; xFlank[i]=0.0; yFlank[i]=0.0;
        //Loop over all flow paths at the i-th step
        for (j=0;j<scroll->flowVec[i].N;j++)
        {
            // The j-th element of the flow vector is a radial flow path
            if (scroll->flowVec[i].flowModel[j]==DRY_GAS_RADIAL_FRICTIONAL_MODEL)
            {
                // Find the irreversibility generated in this flow path E=mdot*(e1-e2)
                xRadial[i]=scroll->theta[i];
                e1=e_m(scroll->Ref,scroll->Liq,scroll->flowVec[i].T_up[j],scroll->flowVec
                    [i].p_up[j],scroll->flowVec[i].xL[j]);
                e2=e_m(scroll->Ref,scroll->Liq,scroll->flowVec[i].T_down[j],scroll->
                    flowVec[i].p_down[j],scroll->flowVec[i].xL[j]);
                yRadial[i]+=fabs(scroll->flowVec[i].mdot[j])*(e1-e2);
            }
            // The j-th element of the flow vector is a flank flow path
            else if (scroll->flowVec[i].flowModel[j]==DRY_GAS_FLANK_FRICTIONAL_MODEL ||
                  scroll->flowVec[i].flowModel[j]==DRY_GAS_FLANK_FLANK_MODEL)
            {
                // Find the irreversibility generated in this flow path E=mdot*(e1-e2)
                xFlank[i]=scroll->theta[i];
                e1=e_m(scroll->Ref,scroll->Liq,scroll->flowVec[i].T_up[j],scroll->flowVec
                    [i].p_up[j],scroll->flowVec[i].xL[j]);
                e2=e_m(scroll->Ref,scroll->Liq,scroll->flowVec[i].T_down[j],scroll->
                    flowVec[i].p_down[j],scroll->flowVec[i].xL[j]);
                yFlank[i]+=fabs(scroll->flowVec[i].mdot[j])*(e1-e2);
            }
            else // Neither flank nor radial leakage
            {
                xFlank[i]=scroll->theta[i];
                yFlank[i]+=0.0;
            }
        }
    }
    // Calculate the average loss rate
    scroll->Losses.leakage_flank=trapz(xFlank,yFlank,scroll->Ntheta)/(2.0*PI);
    scroll->Losses.leakage_radial=trapz(xRadial,yRadial,scroll->Ntheta)/(2.0*PI);
    // Free allocated vectors
    free(xRadial);
    free(yRadial);
    free(xFlank);
```

```
free(yFlank);
// ****************************
// Calculation of suction losses
// ****************************
//Allocate empty temporary vectors
xSuction=(double *)malloc(scroll->Ntheta*sizeof(double));
ySuction=(double *)malloc(scroll->Ntheta*sizeof(double));
// Loop over all points of the revolution
for (i=0;i<scroll->Ntheta;i++)
{
   // If the pressure is below the inlet pressure the extra p-v work is a loss
   if (scroll->p[Is1+NCV*i]<scroll->p[Isuction])
   {
      xSuction[i]=scroll->V[Is1+NCV*i];
      ySuction[i]=scroll->p[Isuction]-scroll->p[Is1+NCV*i];
   }
   // Otherwise it is just "normal" compression work and don't count it
   else
   {
      xSuction[i]=scroll->V[Is1+NCV*i];
      ySuction[i]=0.0;
   }
}
/*
The extra work done for one cycle is W_cycle=int(P*dV),
and the rate of power consumption is W_cycle/(cycle/s)
Multiplied by 2.0 is because there are two symmetric suction chambers
*/
scroll->Losses.suction=2.0*trapz(xSuction,ySuction,scroll->Ntheta)*scroll->omega
    /(2*PI);
// Free allocated vectors
free(xSuction);
free(ySuction);
// ****************************
// Calculation of discharge losses
// ****************************
/*
Approximately, take an effective pressure and volume of the mixed chambers (d1+d2+
    dd) at the discharge angle, compress
it to the discharge pressure adiabatically, then compress at constant pressure to
    the clearance volume.
If the pressure at the discharge angle is above the discharge pressure, then the
    ideal compression process is to take the
merged chamber volume at the discharge pressure to the clearance volume
*/
// First find the index at the discharge angle
i=0;
while (i<scroll->Ntheta)
{
   // if theta brackets the discharge angle
   if ( scroll->theta[i] < theta_d(&(scroll->geo)) && scroll->theta[i+1] > theta_d
       (&(scroll->geo)) )
   {
      // Use the index to the right of the discharge angle and jump out of the
          while() loop
      iD=i+1;
      break;
   }
   i++;
}
//Next find the clearance volume by finding the minimimum of Vddd
i=0; Vcl=99999999;
while (i<scroll->Ntheta)
{
   if ( scroll->V[Iddd+NCV*i] < Vcl &&  scroll->V[Iddd+NCV*i]>0.0)
   {
      Vcl=scroll->V[Iddd+NCV*i];
   }
   i++;
}
V1=scroll->V[Id1+NCV*iD]+scroll->V[Id2+NCV*iD]+scroll->V[Idd+NCV*iD];
//Take the volume-weighted effective pressure of the discharge region
P1=(scroll->V[Id1+NCV*iD]*scroll->p[Id1+NCV*iD]+scroll->V[Id2+NCV*iD]*scroll->p[
    Id2+NCV*iD]+scroll->V[Idd+NCV*iD]*scroll->p[Idd+NCV*iD])/V1;
T1=(scroll->V[Id1+NCV*iD]*scroll->T[Id1+NCV*iD]+scroll->V[Id2+NCV*iD]*scroll->T[
    Id2+NCV*iD]+scroll->V[Idd+NCV*iD]*scroll->T[Idd+NCV*iD])/V1;
xL1=(scroll->V[Id1+NCV*iD]*scroll->xL[Id1+NCV*iD]+scroll->V[Id2+NCV*iD]*scroll->xL
    [Id2+NCV*iD]+scroll->V[Idd+NCV*iD]*scroll->xL[Idd+NCV*iD])/V1;
```

```
    if (P1 < scroll->p[Idischarge])
    {
        //First compress this chamber to the discharge pressure adiabatically
        P2=scroll->p[Idischarge];
        // Mixture ratio of specific heats
        k=kstar_m(scroll->Ref,scroll->Liq,T1,P1,xL1);
        // The volume at the discharge pressure if
        // compressed adiabatically from (V1,P1) to (V2,P2)
        V2=V1*pow(P1/P2,1.0/k);
        // Polytropic compression to P2, then constant pressure to clearance volume
        PV_ideal=(P1*V1-P2*V2)/(1.0-k)-P2*(Vcl-V2);
    }
    else
    {
        PV_ideal=scroll->p[Idischarge]*(Vcl-V1);
    }
    // Allocate integration vectors
    xd1=(double *)malloc(scroll->Ntheta*sizeof(double));
    yd1=(double *)malloc(scroll->Ntheta*sizeof(double));
    xd2=(double *)malloc(scroll->Ntheta*sizeof(double));
    yd2=(double *)malloc(scroll->Ntheta*sizeof(double));
    xdd=(double *)malloc(scroll->Ntheta*sizeof(double));
    ydd=(double *)malloc(scroll->Ntheta*sizeof(double));
    xddd=(double *)malloc(scroll->Ntheta*sizeof(double));
    yddd=(double *)malloc(scroll->Ntheta*sizeof(double));
    // Loop over all points of the revolution
    for (i=0;i<scroll->Ntheta;i++)
    {
        // Load up integration vectors
        xd1[i]=scroll->V[Id1+NCV*i];
        yd1[i]=scroll->p[Id1+NCV*i];
        xd2[i]=scroll->V[Id2+NCV*i];
        yd2[i]=scroll->p[Id2+NCV*i];
        xdd[i]=scroll->V[Idd+NCV*i];
        ydd[i]=scroll->p[Idd+NCV*i];
        xddd[i]=scroll->V[Iddd+NCV*i];
        yddd[i]=scroll->p[Iddd+NCV*i];
    }
    // Calculate the loss term as a sum of the actual PV work minus the ideal PV work
    PV_actual=-(intPV(xd1,yd1,scroll->Ntheta)+intPV(xd2,yd2,scroll->Ntheta)+intPV(xdd,
        ydd,scroll->Ntheta)+intPV(xddd,yddd,scroll->Ntheta));
    scroll->Losses.discharge=(PV_actual-PV_ideal)*scroll->omega/(2.0*PI);
    // Free the vectors
    free(xd1);
    free(yd1);
    free(xd2);
    free(yd2);
    free(xdd);
    free(ydd);
    free(xddd);
    free(yddd);
}
```

FloodProp.h

```
/* File FloodProp.h */
 #ifndef FLOODPROP_H
 #define FLOODPROP_H
    //Define macros for different types of input parameters for EOS
    #define TP 1
    #define Trho 2
    //  ****************************
    //  ***** Gas Variables ********
    //  ****************************
    double cp_A[6], cp_B[6], cp_C[6], cp_D[6], cp_E[6];
    double kg_A[7], kg_B[7], kg_C[7];
    double w[6], Pc[6], Tc[6], MM_g[6];
    double mug_A[7], mug_B[7], mug_C[7];
    //  ****************************
    //  ***** Liq Variables ********
    //  ****************************
    #define NL 8
    double rhol_A[NL], rhol_B[NL], rhol_n[NL], rhol_Tc[NL];
    double kl_A[NL], kl_B[NL], kl_C[NL], kl_D[NL];
```

```
    double cl_A[NL], cl_B[NL], cl_C[NL], cl_D[NL];
    double mul_A[NL], mul_B[NL], mul_C[NL], mul_D[NL];
    double MM_l[NL];
    static int I_N2=0, I_He=1, I_Ne=2, I_Ar=3, I_Kr=4, I_Xe=5, I_CO2=6;
    static int I_Methanol=0, I_Ethanol=1, I_Propanol=2, I_Butanol=3, I_Water=4, I_NH3
        =5, I_Zerol=6,I_POE=7;
    double hm2(double T, double P, double xL);
    double cK_e(double v_l, double v_g, double x, double w, double flag);
    double cv_e(double v_l, double v_g, double K_e, double x, double w, double flag);
    double R(char *Gas);
    double rho_l(char *Liq, double T);
    double rho_m(char *Gas, char *Liq, double T, double P, double xL);
    double u_l(char *Liq, double T);
    double h_g(char *Gas,double T, double P);
    double u_m(char *Gas, char *Liq, double T, double P, double xL);
    double h_m(char *Gas, char *Liq, double T, double P, double xL);
    double c_v(char *Gas, double T, double P);
    double c_l(char *Liq, double T);
    double dudT_m(char *Gas, char *Liq, double T, double P, double xL);
    double dudP_m(char *Gas, char *Liq, double T, double P, double xL);
    double dudxL_m(char *Gas, char *Liq, double T, double P, double xL);
    double drhodP_m(char *Gas, char *Liq, double T, double P, double xL);
    double drhodT_m(char *Gas, char *Liq, double T, double P, double xL);
    double drhodxL_m(char *Gas, char *Liq, double T, double P, double xL);
    double cp_mix(char *Gas, char *Liq, double T, double P, double xL);
    double mu_mix(char *Gas, char *Liq, double T, double p, double xL);
    double k_mix(char *Gas, char *Liq, double T, double P, double xL);
    double Pr_mix(char *Gas, char *Liq, double T,double P,double xL);
    double s_m(char *Gas, char *Liq, double T, double P, double xL);
    double e_m(char *Gas, char *Liq, double T, double P, double xL);
    double dvdT_m(char *Gas, char *Liq, double T, double P, double xL);
    double dvdP_m(char *Gas, char *Liq, double T, double P, double xL);
    int getIndex(char *Fluid);
    double mu_l(char *Liq, double T);
    double mu_g(char *Gas, double T, double p);
    double k_l(char *Liq, double T);
    double k_g(char *Gas, double T, double p);
    double s_l(char *Liq, double T);
    double s_g(char *Gas, double T, double P);
    double VoidFrac(char *Gas, char *Liq, double T, double P, double xL);
    double c_p(char *Gas, double T, double P);
    double kstar_m(char *Gas, char *Liq, double T, double P, double xL);
    double rho_g(char *Gas, double T, double P);
    double u_g(char *Gas, double T, double P);
    double T_hp(char *Gas, char *Liq, double h, double p, double xL, double T_guess);
    double T_Up(char *Gas, char *Liq, double U, double p, double xL, double V, double
        T_guess);
    double h_sp(char *Gas, char *Liq, double s, double p, double xL, double T_guess);
    double T_sp(char *Gas, char *Liq, double s, double p, double xL, double T_guess);
    double p_Trho(char *Gas, char *Liq, double rho, double T, double xL, double
        p_guess);
    /*double c_g(char *Gas, double T, double p)*/;
    int isNAN_FP(double x);
    int isINFINITY_FP(double x);
    void setGas();
    void setLiq();
    double dpdT_const_v(char *Gas, char *Liq, double T, double p1, double xL);
#endif
```

FloodProp.c

```
#ifndef __GNUC__
#define _CRTDBG_MAP_ALLOC
#include <stdlib.h>
#include <crtdbg.h>
#endif
#include "math.h"
#include "stdio.h"
#include <string.h>
#include "FloodProp.h"
#include "R744.h"
#include "R404A.h"
#include "R134a.h"
#include "R410A.h"
#include "Nitrogen.h"
#include "PropMacros.h"
```

```
//When adding gas, make sure to increase
// the length of coefficient vectors

int isNAN_FP(double x)
{
   // recommendation from http://www.devx.com/tips/Tip/42853
   return x != x;
}
int isINFINITY_FP(double x)
{
   // recommendation from http://www.devx.com/tips/Tip/42853
   if ((x == x) && ((x - x) != 0.0))
      return 1;//return (x < 0.0 ? -1 : 1); // This will tell you whether positive or
          negative infinity
   else
      return 0;
}
int getIndex(char *Fluid)
{
    int I;
    if (!strcmp(Fluid,"N2")) {I=I_N2;}
    if (!strcmp(Fluid,"He")) {I=I_He;}
    if (!strcmp(Fluid,"Ne")) {I=I_Ne;}
    if (!strcmp(Fluid,"Ar")) {I=I_Ar;}
    if (!strcmp(Fluid,"Kr")) {I=I_Kr;}
    if (!strcmp(Fluid,"Xe")) {I=I_Xe;}
    if (!strcmp(Fluid,"CO2")) {I=I_CO2;}
    if (!strcmp(Fluid,"Methanol")) { I=I_Methanol;}
    if (!strcmp(Fluid,"Ethanol")) { I=I_Ethanol;}
    if (!strcmp(Fluid,"Propanol")) {I=I_Propanol;}
    if (!strcmp(Fluid,"Butanol")) {I=I_Butanol;}
    if (!strcmp(Fluid,"Water")) {I=I_Water;}
    if (!strcmp(Fluid,"NH3")) {I=I_NH3;}
    if (!strcmp(Fluid,"Zerol")) {I=I_Zerol;}
   if (!strcmp(Fluid,"POE")) {I=I_POE;}
    return I;
}
// // ***********************************************************************
// // ************************  CONSTANTS  ******************************
// // ***********************************************************************
void setGas()
{
    //Constants for Molar specific heat
    //cp in [kJ/kg-K]
        cp_A[I_N2]=29.342;
        cp_B[I_N2]=-0.0035395;
        cp_C[I_N2]=0.000010076;
        cp_D[I_N2]=-4.3116E-09;
        cp_E[I_N2]=2.5935E-13;
        cp_A[I_He]=20.786;
        cp_B[I_He]=0.0;
        cp_C[I_He]=0.0;
        cp_D[I_He]=0.0;
        cp_E[I_He]=0.0;
        cp_A[I_Ne]=20.786;
        cp_B[I_Ne]=0.0;
        cp_C[I_Ne]=0.0;
        cp_D[I_Ne]=0.0;
        cp_E[I_Ne]=0.0;
        cp_A[I_Ar]=20.786;
        cp_B[I_Ar]=0.0;
        cp_C[I_Ar]=0.0;
        cp_D[I_Ar]=0.0;
        cp_E[I_Ar]=0.0;
        cp_A[I_Kr]=20.786;
        cp_B[I_Kr]=0.0;
        cp_C[I_Kr]=0.0;
        cp_D[I_Kr]=0.0;
        cp_E[I_Kr]=0.0;
        cp_A[I_Xe]=20.786;
        cp_B[I_Xe]=0.0;
        cp_C[I_Xe]=0.0;
        cp_D[I_Xe]=0.0;
```

```
        cp_E[I_Xe]=0.0;
    //Constants for Molar specific heat
    //cp in [kJ/kg-K]
    // Thermal Conductivity
    // Units of [W/m-K]
        kg_A[I_N2]=0.00309;     kg_B[I_N2]=7.593e-5;        kg_C[I_N2]=-1.1014e-8;
        kg_A[I_He]=0.05516;     kg_B[I_He]=0.0003254;       kg_C[I_He]=-2.2723E-08;
        kg_A[I_Ne]=0.01379;     kg_B[I_Ne]=0.00012156;      kg_C[I_Ne]=-2.359E-08;
        kg_A[I_Ar]=0.00548;     kg_B[I_Ar]=0.000043869;     kg_C[I_Ar]=-6.8141E-09;
        kg_A[I_Kr]=0.00168;     kg_B[I_Kr]=0.000027493;     kg_C[I_Kr]=-4.7254E-09;
        kg_A[I_Xe]=0.00034;     kg_B[I_Xe]=0.000018809;     kg_C[I_Xe]=-3.0072E-09;
        kg_A[I_CO2]=-0.012;      kg_B[I_CO2]=0.00010208;     kg_C[I_CO2]=-2.2403E-08;
    //Thermal Conductivity
    //Units of [W/m-K]
    //Accentric Factor For Gas
    //Dimensionless
        w[I_N2]=0.04;
        w[I_He]=-0.39;
        w[I_Ne]=-0.04;
        w[I_Ar]=0.0;
        w[I_Kr]=0.0;
        w[I_Xe]=0.0;
    //Accentric Factor For Gas
    //Dimensionless
    //Critical Temp For Gas
    //Units of [K]
        Tc[I_N2]=126.1;
        Tc[I_He]=5.2;
        Tc[I_Ne]=44.4;
        Tc[I_Ar]=150.86;
        Tc[I_Kr]=209.35;
        Tc[I_Xe]=289.74;
    //Critical Temp For Gas
    //Units of [K]
    //Critical Pressure For Gas
    //Units of [kPa]
        Pc[I_N2]=3394.0;
        Pc[I_He]=228.0;
        Pc[I_Ne]=2653.0;
        Pc[I_Ar]=4898.0;
        Pc[I_Kr]=5502.0;
        Pc[I_Xe]=5840.0;
    //Critical Pressure For Gas
    //Units of [kPa]
    //Viscosity Constants of Gas
    //Viscosity has units of micro-Poise
        mug_A[I_Ar]=44.997;     mug_B[I_Ar]=0.63892;  mug_C[I_Ar]=-0.00012455;
        mug_A[I_He]=71.094;     mug_B[I_He]=0.443;      mug_C[I_He]=-0.0000518;
        mug_A[I_Kr]=31.096;     mug_B[I_Kr]=0.798;      mug_C[I_Kr]=-0.000179;
        mug_A[I_Ne]=102.964;  mug_B[I_Ne]=0.746;      mug_C[I_Ne]=-0.000136;
        mug_A[I_N2]=42.606;     mug_B[I_N2]=0.475;      mug_C[I_N2]=-0.0000988;
        mug_A[I_Xe]=7.386;      mug_B[I_Xe]=0.787;      mug_C[I_Xe]=-0.000151;
        mug_A[I_CO2]=11.811;    mug_B[I_CO2]=0.49838;   mug_C[I_CO2]=-0.00010851;
    //Viscosity Constants of Gas
    //Viscosity has units of micro-Poise
    //Mole Mass of Gas
    //Units of kg/kmol
        MM_g[I_Ar]=39.948;
        MM_g[I_He]=4.003;
        MM_g[I_Kr]=83.8;
        MM_g[I_Ne]=20.18;
        MM_g[I_N2]=28.013;
        MM_g[I_Xe]=131.29;
    //Mole Mass of Gas
    //Units of kg/kmol
}
void setLiq()
{
    //Constants for Molar specific heat
    //cp in [kJ/kg-K]
        cl_A[I_Methanol]=40.152;
        cl_B[I_Methanol]=0.31046;
        cl_C[I_Methanol]=-0.0010291;
        cl_D[I_Methanol]=1.4598E-06;
        cl_A[I_Ethanol]=59.342;
        cl_B[I_Ethanol]=0.36358;
```

```
    cl_C[I_Ethanol]=-0.0012164;
    cl_D[I_Ethanol]=0.000001803;

    cl_A[I_Propanol]=72.525;
    cl_B[I_Propanol]=0.79553;
    cl_C[I_Propanol]=-0.002633;
    cl_D[I_Propanol]=3.6498E-06;

    cl_A[I_Butanol]=95.037;
    cl_B[I_Butanol]=0.56593;
    cl_C[I_Butanol]=-0.0018256;
    cl_D[I_Butanol]=2.6675E-06;

    cl_A[I_Water]=92.053;
    cl_B[I_Water]=-0.039953;
    cl_C[I_Water]=-0.00021103;
    cl_D[I_Water]=5.3469E-07;

    cl_A[I_NH3]=-182.157;
    cl_B[I_NH3]=3.3618;
    cl_C[I_NH3]=-0.014398;
    cl_D[I_NH3]=0.000020371;

    cl_A[I_Zerol]=337.116/1000;
    cl_B[I_Zerol]=5.186/1000;
    cl_C[I_Zerol]=0.0;
    cl_D[I_Zerol]=0.0;

  cl_A[I_POE]=1.8; // From Booser page 16
    cl_B[I_POE]=0.0;
    cl_C[I_POE]=0.0;
    cl_D[I_POE]=0.0;

//Constants for Molar specific heat
//cp in [kJ/kg-K]

// Thermal Conductivity
// Units of [W/m-K]
    kl_A[I_Methanol]=-1.1793;   kl_B[I_Methanol]=0.6191;   kl_C[I_Methanol
        ]=512.58;
    kl_A[I_Ethanol]=-1.3172; kl_B[I_Ethanol]=0.6987;     kl_C[I_Ethanol]=516.25;
    kl_A[I_Propanol]=-1.3721;   kl_B[I_Propanol]=0.658;     kl_C[I_Propanol
        ]=508.31;
    kl_A[I_Butanol]=-1.4633; kl_B[I_Butanol]=0.7473;     kl_C[I_Butanol]=536.01;
    kl_A[I_Water]=-0.2758;      kl_B[I_Water]=0.004612;     kl_C[I_Water]=-5.5391
        E-06;
    kl_A[I_NH3]=1.1606;         kl_B[I_NH3]=-0.002284;      kl_C[I_NH3]=3.1245E
        -18;
    kl_A[I_Zerol]=0.1700;       kl_B[I_Zerol]=0;            kl_C[I_Zerol]=0;
  kl_A[I_POE]=0.147;          kl_B[I_POE]=0;              kl_C[I_POE]=0;
// Thermal Conductivity
// Units of [W/m-K]

//Viscosity Constants of Liquid
//Viscosity has units of centi-Poise
    mul_A[I_Methanol]=-9.0562;  mul_B[I_Methanol]=1254.2;  mul_C[I_Methanol
        ]=0.022383;     mul_D[I_Methanol]=-0.000023538;
    mul_A[I_Ethanol]=-6.4406;   mul_B[I_Ethanol]=1117.6;   mul_C[I_Ethanol
        ]=0.013721;      mul_D[I_Ethanol]=-0.000015465;
    mul_A[I_Propanol]=-0.7009;  mul_B[I_Propanol]=841.5;   mul_C[I_Propanol
        ]=-0.0086068; mul_D[I_Propanol]=8.2964E-06;
    mul_A[I_Butanol]=-20.6736;  mul_B[I_Butanol]=3549.3;   mul_C[I_Butanol
        ]=0.040352;      mul_D[I_Butanol]=-0.000030937;
    mul_A[I_Water]=-10.2158; mul_B[I_Water]=1792.5;      mul_C[I_Water]=0.01773;
                mul_D[I_Water]=-0.000012631;
    mul_A[I_NH3]=-8.591;        mul_B[I_NH3]=876.4;         mul_C[I_NH3]=0.02681;
                    mul_D[I_NH3]=-0.00003612;
    mul_A[I_Zerol]=0.0102;      mul_B[I_Zerol]=0;           mul_C[I_Zerol]=0;
                      mul_D[I_Zerol]=0;
  mul_A[I_POE]=0.0102;        mul_B[I_POE]=0;             mul_C[I_POE]=0;
                    mul_D[I_POE]=0;
//Viscosity Constants of Liquid
//Viscosity has units of centi-Poise

//Density Constants of Liquid
//Density has units of kg/m^3
    rhol_A[I_Methanol]=0.27197; rhol_B[I_Methanol]=0.27192;   rhol_n[I_Methanol
        ]=0.2331; rhol_Tc[I_Methanol]=512.58;
    rhol_A[I_Ethanol]=0.2657;   rhol_B[I_Ethanol]=0.26395; rhol_n[I_Ethanol
        ]=0.2367;  rhol_Tc[I_Ethanol]=516.25;
    rhol_A[I_Propanol]=0.26785; rhol_B[I_Propanol]=0.26475;   rhol_n[I_Propanol
        ]=0.243;  rhol_Tc[I_Propanol]=508.31;
    rhol_A[I_Butanol]=0.27343;  rhol_B[I_Butanol]=0.2635;  rhol_n[I_Butanol
        ]=0.2604;  rhol_Tc[I_Butanol]=536.01;
    rhol_A[I_Water]=0.3471;     rhol_B[I_Water]=0.274;      rhol_n[I_Water
        ]=0.28571;  rhol_Tc[I_Water]=647.13;
```

```
        rhol_A[I_NH3]=0.23689;      rhol_B[I_NH3]=0.25471;      rhol_n[I_NH3]=0.2887;
                    rhol_Tc[I_NH3]=405.65;
    //Density Constants of Liquid
    //Density has units of kg/m^3
    //Mole Mass of Liquid
    //Units of kg/kmol
        MM_l[I_Methanol]=32.04;
        MM_l[I_Ethanol]=46.06844;
        MM_l[I_Propanol]=60.10;
        MM_l[I_Butanol]=74.1216;
        MM_l[I_Water]=18.0153;
        MM_l[I_NH3]=17.0306;
        MM_l[I_Zerol]=1;  //Dummy value since values for Zerol are given
                          // in terms of kJ/kg and dont need conversion
      MM_l[I_POE]=1;    //Dummy value since values for POE are given
                          // in terms of kJ/kg and dont need conversion
    //Mole Mass of Liquid
    //Units of kg/kmol
}
// // ***********************************************************************
// // ************************  FUNCTIONS  ******************************
// // ***********************************************************************
double cK_e(double v_l, double v_g, double x, double w, double flag)
{
    // Equation taken from page 43, equation 4.51 from Chisholm for
    // liquid entrainment in gas.  Value of w 0.4 is recommended from text
    double KE;
    if (x==0 || x==1)
    {
        return 1;
    }
    if (flag>0.9 && flag<1.1)
    {
        KE=w+(1.0-w)*sqrt((v_g/v_l+w*(1-x)/x)/(1+w*(1-x)/x));
    }
    if (flag>1.9 && flag<2.1)
    {
        // Chisholms sqrt(vh/vl)
        KE=sqrt(1.0+x*(v_g/v_l-1.0));
    }
    if (flag>2.9 && flag<3.1)
    {
        KE=pow(v_g/v_l,0.25*.28);
    }
    return KE;
}
double cv_e(double v_l, double v_g, double K_e, double x, double w, double flag)
{
    double ve;
    double Kc;
    if (flag>0.9 && flag<1.1)
    {
        // using 5.48 and 5.49 from Chisholm
        // if w=0, separated flow results ( flag==2 )
        // if w=1, homogeneous flow results ( flag == 5)
        // So basically this form is general and captures all possibilities, sep, hom
            , or entrained
        // should use this one
        //      (     [(1-w)^2] ) -1
        // Kc= ( w+ [-------] )
        //      (     [ K_e-w ] )
        Kc=1.0/(w+((1.0-w)*(1.0-w))/(K_e-w));
        ve=(x*v_g+K_e*(1.0-x)*v_l)*(x+(1.0-x)/Kc);
    }
    if (flag>1.9 && flag<2.1)
    {
        // Equation 5.13 from Chisholm for separated flow
        ve=(x*v_g+K_e*(1.0-x)*v_l)*(x+(1.0-x)/K_e);
    }
    if (flag>2.9 && flag<3.1)
    {
        // Equation 2.48 from Chisholm
        ve=(1+w*(1-x)/x*v_l/v_g)/(1+w*(1-x)/x)*v_g;
    }
    if (flag>3.9 && flag<4.1)
    {
        // using 15 from Morris
```

```
        // If going to use this formula, must change pow() to multiply
        ve=(x*v_g+K_e*(1.0-x)*v_l)*(x+(1.0-x)/K_e*(1+pow(K_e-1,2.0)/(pow(v_g/v_l,0.5)
            -1)));
    }
    if (flag>4.9 && flag<5.1)
    {
        // homogenous
        ve=(x*v_g+(1.0-x)*v_l);
    }
    return ve;
}
double R(char *Gas)
{
    // output in kJ/kg-K
   int ii;
   if (!strcmp(Gas,"CO2"))
      return 8.31447215/MM_R744();
   if (!strcmp(Gas,"R410A"))
      return 8.31447215/MM_R410A();
   if (!strcmp(Gas,"R404a"))
      return 8.31447215/MM_R404A();
   if (!strcmp(Gas,"R134a"))
      return 8.31447215/MM_R134a();
   if (!strcmp(Gas,"Nitrogen") || !strcmp(Gas,"N2"))
      return 8.31447215/MM_Nitrogen();
    setGas();
    ii=getIndex(Gas);
    return 8.31447215/MM_g[ii];
}
double rho_l(char *Liq, double T)
{
    // T in K
    // rho in kg/m^3
    double rhoL;
    int ii;
    setLiq();
    ii=getIndex(Liq);
    if (!strcmp(Liq,"Zerol"))
        rhoL=-.6670*T +1050.865;
   else if (!strcmp(Liq,"POE"))
   {
      // Based on 3MAF POE oil data provided by Emerson Climate
      rhoL=(-0.00074351165052847*(T-273.15)+0.992439584365375)*1000;
   }
    else
    {
        //Given by the form rho=A/B^((1-T/Tc)^n)
        rhoL=rhol_A[ii]/pow(rhol_B[ii],pow(1-T/rhol_Tc[ii],rhol_n[ii]))*1000;
    }
    return rhoL;
}
double rho_g(char *Gas, double T, double P)
{
    // input in K, [-]
    // output in kg/m^3
    if (!strcmp(Gas,"CO2"))
      return rho_R744(T,P,TYPE_TP);
   if (!strcmp(Gas,"R410A"))
      return rho_R410A(T,P,TYPE_TP);
   if (!strcmp(Gas,"R404A"))
      return rho_R404A(T,P,TYPE_TP);
   if (!strcmp(Gas,"R134a"))
      return rho_R134a(T,P,TYPE_TP); // necessary to avoid recalculation
   if (!strcmp(Gas,"Nitrogen"))
      return rho_Nitrogen(T,P,TYPE_TP); // necessary to avoid recalculation
    return P/(R(Gas)*T);
}
double rho_m(char *Gas, char *Liq, double T, double P, double xL)
{
    // input in K, kPa, [-]
    // output in kg/m^3
    double vG, vL, x,rhom;
    if (xL==0)
    {
        return rho_g(Gas,T,P);
    }
    if (xL==1)
```

```
    {
        return rho_l(Liq,T);
    }
    vG=1/rho_g(Gas,T,P);
    vL=1/rho_l(Liq,T);
    x=1-xL;
   rhom=1/(vG*x+vL*(1-x));
   if (isNAN_FP(rhom))
      printf("rhom is NaN");
   if (isINFINITY_FP(rhom))
      printf("rhom is Infinite");
    return rhom;
}
double h_g(char *Gas, double T, double P)
{
    // input in K,kPa
    // output in kJ/kg
    int ii;
   if (!strcmp(Gas,"CO2"))
      return h_R744(T,P,TYPE_TP);
   if (!strcmp(Gas,"R410A"))
      return h_R410A(T,P,TYPE_TP);
   if (!strcmp(Gas,"R404A"))
      return h_R404A(T,P,TYPE_TP);
   if (!strcmp(Gas,"R134a"))
      return h_R134a(T,P,TYPE_TP);
   if (!strcmp(Gas,"Nitrogen"))
      return h_Nitrogen(T,P,TYPE_TP);
    setGas();
   ii=getIndex(Gas);
   return (cp_A[ii]*(T-298.15) + cp_B[ii]/2.0*(T*T-298.15*298.15) + cp_C[ii]/3.0*(T*T
       *T-298.15*298.15*298.15) + cp_D[ii]/4.0*(T*T*T*T-298.15*298.15*298.15*298.15)
       + cp_E[ii]/5.0*(T*T*T*T*T-298.15*298.15*298.15*298.15*298.15))/MM_g[ii];
}
double u_g(char *Gas, double T, double P)
{
    // input in K,kPa
    // output in kJ/kg
    if (!strcmp(Gas,"CO2"))
      return u_R744(T,P,TYPE_TP);
   if (!strcmp(Gas,"R410A"))
      return u_R410A(T,P,TYPE_TP);
   if (!strcmp(Gas,"R404A"))
      return u_R404A(T,P,TYPE_TP);
   if (!strcmp(Gas,"R134a"))
      return u_R134a(T,P,TYPE_TP);
   if (!strcmp(Gas,"Nitrogen"))
      return u_Nitrogen(T,P,TYPE_TP);
   return h_g(Gas,T,P)-R(Gas)*T;
}
double s_g(char *Gas, double T, double P)
{
    // input in K
    // output in kJ/kg-K
    int ii;
    if (!strcmp(Gas,"CO2"))
      return s_R744(T,P,TYPE_TP);
   if (!strcmp(Gas,"R410A"))
      return s_R410A(T,P,TYPE_TP);
   if (!strcmp(Gas,"R404A"))
      return s_R404A(T,P,TYPE_TP);
   if (!strcmp(Gas,"R134a"))
      return s_R134a(T,P,TYPE_TP);
   if (!strcmp(Gas,"Nitrogen"))
      return s_Nitrogen(T,P,TYPE_TP);
    setGas();
    ii=getIndex(Gas);
    return (cp_A[ii]*log(T/298.15) + cp_B[ii]*(T-298.15) + cp_C[ii]/2.0*(T*T
        -298.15*298.15) + cp_D[ii]/3.0*(T*T*T-298.15*298.15*298.15))/MM_g[ii]-R(Gas)*
        log(P/101.325);
}
double u_m(char *Gas, char *Liq, double T, double P, double xL)
{
    // input in K, [-]
    // output in kJ/kg
```

```
    double uG, uL,um;
        uL=u_l(Liq,T);
        uG=u_g(Gas,T,P);
      um=xL*uL+(1-xL)*uG;
   if (isNAN_FP(um))
      printf("um is NaN");
   if (isINFINITY_FP(um))
      printf("um is Infinite");
    return um;
}
double h_m(char *Gas, char *Liq, double T, double P, double xL)
{
    // input in K, [-]
    // output in kJ/kg
    double hG,hL,hm;
    hG=h_g(Gas,T,P);
    hL=u_l(Liq,T)+(P-101.325)/rho_l(Liq,T);
   hm=xL*hL+(1-xL)*hG;
   if (isNAN_FP(hm))
      printf("hm is NaN");
   if (isINFINITY_FP(hm))
      printf("hm is Infinite");
    return hm;
}
double c_v(char *Gas, double T, double P)
{
    // input in K, [-]
    // output in kJ/kg-K
    int ii;
   if (!strcmp(Gas,"CO2"))
      return cv_R744(T,P,TYPE_TP);
   if (!strcmp(Gas,"R410A"))
      return cv_R410A(T,P,TYPE_TP);
   if (!strcmp(Gas,"R404A"))
      return cv_R404A(T,P,TYPE_TP);
   if (!strcmp(Gas,"R134a"))
      return cv_R134a(T,P,TYPE_TP);
   if (!strcmp(Gas,"Nitrogen"))
      return cv_Nitrogen(T,P,TYPE_TP);
    setGas();
    ii=getIndex(Gas);
    return (cp_A[ii] + cp_B[ii]*T + cp_C[ii]*T*T + cp_D[ii]*T*T*T + cp_E[ii]*T*T*T*T)
        /MM_g[ii]-R(Gas);
}
double c_p(char *Gas, double T, double P)
{
    // input in K, [-]
    // output in kJ/kg-K
    int ii;
   if (!strcmp(Gas,"CO2"))
      return cp_R744(T,P,TYPE_TP);
   if (!strcmp(Gas,"R410A"))
      return cp_R410A(T,P,TYPE_TP);
   if (!strcmp(Gas,"R404A"))
      return cp_R404A(T,P,TYPE_TP);
   if (!strcmp(Gas,"R134a"))
      return cp_R134a(T,P,TYPE_TP);
   if (!strcmp(Gas,"Nitrogen"))
      return cp_Nitrogen(T,P,TYPE_TP);
    setGas();
    ii=getIndex(Gas);
    return (cp_A[ii] + cp_B[ii]*T + cp_C[ii]*T*T + cp_D[ii]*T*T*T + cp_E[ii]*T*T*T*T)
        /MM_g[ii];
}
double c_l(char *Liq, double T)
{
    // input in K, [-]
    // output in kJ/kg
    int ii;
    setLiq();
    ii=getIndex(Liq);
    return (cl_A[ii] + cl_B[ii]*T + cl_C[ii]*T*T + cl_D[ii]*T*T*T)/MM_l[ii];
}
double u_l(char *Liq, double T)
{
    // input in K
```

```
    // output in kJ/kg
    int ii;
    setLiq();
    ii=getIndex(Liq);
    return (cl_A[ii]*(T-298.15) + cl_B[ii]/2.0*(T*T-298.15*298.15) + cl_C[ii]/3.0*(T*
        T*T-298.15*298.15*298.15) + cl_D[ii]/4.0*(T*T*T*T
        -298.15*298.15*298.15*298.15))/MM_l[ii];
}
double s_l(char *Liq, double T)
{
    // input in K
    // output in kJ/kg-K
    int ii;
    setLiq();
    ii=getIndex(Liq);
    return (cl_A[ii]*log(T/298.15) + cl_B[ii]*(T-298.15) + cl_C[ii]/2.0*(T*T
        -298.15*298.15) + cl_D[ii]/3.0*(T*T*T-298.15*298.15*298.15))/MM_l[ii];
}
double k_l(char *Liq, double T)
{
    // input in K
    // output in kW/m-K
    int ii;
    double kL;
    setLiq();
    ii=getIndex(Liq);
    if (!strcmp(Liq,"NH3") || !strcmp(Liq,"Water") || !strcmp(Liq,"Zerol"))
    {
        kL=(kl_A[ii] + kl_B[ii]*T + kl_C[ii]*T*T)/1000.0;
    }
    else
    {
        kL=(pow(10.0,kl_A[ii] + kl_B[ii]*pow(1-T/kl_C[ii],2.0/7.0)))/1000.0;
    }
    return kL;
}
double mu_l(char *Liq, double T)
{
    // input in K
    // output in cP --> Pa-s
    int ii;
    double muL,mu_cSt;
    if (!strcmp(Liq,"Zerol"))
    {
        muL=-0.000122996*T+0.048002276;
    }
   else if (!strcmp(Liq,"POE"))
   {
      // Based on 3MAF POE oil data provided by Emerson Climate
      mu_cSt= 0.0002389593*log(T)*log(T) - 0.1927238779*log(T) + 40.3718884485;
      // From cSt to m^s, multiply by 1e-6, then multiply by density
      muL=mu_cSt*1e-6*rho_l(Liq,T);
   }
    else
    {
      setLiq();
      ii=getIndex(Liq);
        muL=pow(10.0,mul_A[ii] + mul_B[ii]/T + mul_C[ii]*T + mul_D[ii]*T*T)/1.0e3;
    }
    return muL;
}
double k_g(char *Gas, double T, double p)
{
    // input in K
    // output in kW/m-K
    int ii;
   if (!strcmp(Gas,"R134a"))
      return k_R134a(T,p,TYPE_TP);
   //if (!strcmp(Gas,"CO2") || !strcmp(Gas,"R744"))
//     return k_R744(T,p,TYPE_TP);
   if (!strcmp(Gas,"R410A"))
      return k_R410A(T,p,TYPE_TP);
    setGas();
    ii=getIndex(Gas);
    return (kg_A[ii] + kg_B[ii]*T + kg_C[ii]*T*T)/1000.0;
}
double mu_g(char *Gas, double T, double p)
```

```
{
    // input in K
    // output in microP --> Pa-s
    int ii;
   if (!strcmp(Gas,"R134a"))
      return visc_R134a(T,p,TYPE_TP);
   if (!strcmp(Gas,"CO2") || !strcmp(Gas,"R744"))
        return visc_R744(T,p,TYPE_TP);
   if (!strcmp(Gas,"R410A"))
        return visc_R410A(T,p,TYPE_TP);
    setGas();
    ii=getIndex(Gas);
    return (mug_A[ii] + mug_B[ii]*T + mug_C[ii]*T*T)/1.0e7;
}
double dudT_m(char *Gas, char *Liq, double T, double P, double xL)
{
    double delta=.001;
    return (u_m(Gas,Liq,T+delta,P,xL)-u_m(Gas,Liq,T,P,xL))/delta;
}
double dudP_m(char *Gas, char *Liq, double T, double P, double xL)
{
    double delta=.001;
    return (u_m(Gas,Liq,T,P+delta,xL)-u_m(Gas,Liq,T,P,xL))/delta;
}
double dudxL_m(char *Gas, char *Liq, double T, double P, double xL)
{
    double delta=.001;
    return (u_m(Gas,Liq,T,P,xL+delta)-u_m(Gas,Liq,T,P,xL))/delta;
}
double drhodP_m(char *Gas, char *Liq, double T, double P, double xL)
{
    double delta=0.001;
    return (rho_m(Gas, Liq,T,P+delta,xL)-rho_m(Gas, Liq,T,P,xL))/delta;
}
double drhodT_m(char *Gas, char *Liq, double T, double P, double xL)
{
    double delta=0.001;
    return (rho_m(Gas, Liq,T+delta,P,xL)-rho_m(Gas, Liq,T,P,xL))/delta;
}
double drhodxL_m(char *Gas, char *Liq, double T, double P, double xL)
{
    double delta=.001;
    return (rho_m(Gas, Liq,T,P,xL+delta)-rho_m(Gas, Liq,T,P,xL))/delta;
}
double dvdT_m(char *Gas, char *Liq, double T, double P, double xL)
{
    double delta=.001;
    return (1/rho_m(Gas, Liq,T+delta,P,xL)-1/rho_m(Gas, Liq,T,P,xL))/delta;
}
double dvdP_m(char *Gas, char *Liq, double T, double P, double xL)
{
    double delta=.001;
    return (1/rho_m(Gas, Liq,T,P+delta,xL)-1/rho_m(Gas, Liq,T,P,xL))/delta;
}
double cp_mix(char *Gas, char *Liq, double T, double P, double xL)
{
    // input in K,[-]
    // output in kJ/kg
   double cpm;
   cpm=xL*c_l(Liq,T)+(1-xL)*c_p(Gas,T,P);
   if (isNAN_FP(cpm))
      printf("cpm is NaN");
   if (isINFINITY_FP(cpm))
      printf("cpm is Infinite");
    return cpm;
}
double mu_mix(char *Gas, char *Liq, double T, double p, double xL)
{
    double muL, muG,mum;
    muL=mu_l(Liq,T); //kg/m-s
    muG=mu_g(Gas,T,p);
   mum=1/(xL/muL+(1-xL)/muG);
   if (isNAN_FP(mum))
      printf("mum is NaN");
   if (isINFINITY_FP(mum))
      printf("mum is Infinite");
```

```
    return mum;
}
double k_mix(char *Gas, char *Liq, double T, double P, double xL)
{
    //input in K,kPa, [-]
    //output in kW/m-K
    double kL, kG, VF,km;
    kL=k_l(Liq,T);
    kG=k_g(Gas,T,P);
    VF=VoidFrac(Gas,Liq,T,P,xL);
   km=(1-VF)*kL+VF*kG;
   if (isNAN_FP(km))
      printf("km is NaN");
   if (isINFINITY_FP(km))
      printf("km is Infinite");
    return km;
}
double VoidFrac(char *Gas, char *Liq, double T, double P, double xL)
{
    //input in K, kPa, [-]
    // output in [-]
    double rhoL, rhoG,VF;
    rhoL=rho_l(Liq,T);
    rhoG=rho_g(Gas,T,P);
   VF= ((1-xL)/rhoG)/((1-xL)/rhoG+xL/rhoL);
   if (isNAN_FP(VF))
      printf("VF is NaN");
   if (isINFINITY_FP(VF))
      printf("VF is Infinite");
    return VF;
}
double Pr_mix(char *Gas, char *Liq, double T, double P, double xL)
{
   double cpm;
   //double cp,mu,k;
   //cp=cp_mix(Gas,Liq,T,P,xL);
   //mu=mu_mix(Gas,Liq,T,P,xL);
   //k=k_mix(Gas,Liq,T,P,xL);
   cpm=cp_mix(Gas,Liq,T,P,xL)*mu_mix(Gas,Liq,T,P,xL)/k_mix(Gas,Liq,T,P,xL);
   if (isNAN_FP(cpm))
      printf("cpm is NaN");
   if (isINFINITY_FP(cpm))
      printf("cpm is Infinite");
    return cpm;
}
double kstar_m(char *Gas, char *Liq, double T, double P,double xL)
{
   double kstarm;
   //double cp,cv,cl;
   //cp=c_p(Gas,T,P);
   //cv=c_v(Gas,T,P);
   //cl=c_l(Liq,T);
   kstarm=((1-xL)*c_p(Gas,T,P)+xL*c_l(Liq,T))/((1-xL)*c_v(Gas,T,P)+xL*c_l(Liq,T));
   if (isNAN_FP(kstarm))
      printf("kstarm is NaN");
   if (isINFINITY_FP(kstarm))
      printf("kstarm is Infinite");
    return kstarm;
}
double s_m(char *Gas, char *Liq, double T, double P, double xL)
{
//      input in K, kPa, [-]
//      output in kJ/kg-K
    double sG,sL,sm;
    sG=s_g(Gas,T,P);
    sL=s_l(Liq,T);
   sm=xL*sL+(1-xL)*sG;
   if (isNAN_FP(sm))
      printf("sm is NaN");
   if (isINFINITY_FP(sm))
      printf("sm is Infinite");
    return sm;
}
double e_m(char *Gas, char *Liq, double T, double P, double xL)
{
//      input in K, kPa, [-]
//      output in kJ/kg
```

```
    double T0=300,P0=100;
    return (h_m(Gas,Liq,T,P,xL)-h_m(Gas,Liq,T0,P0,xL))-T0*(s_m(Gas,Liq,T,P,xL)-s_m(
        Gas,Liq,T0,P0,xL));
}
double T_hp(char *Gas, char *Liq, double h, double p, double xL, double T_guess)
{
   double x1,x2,x3,y1,y2,eps,change,f,T;
   int iter;
   eps=1e-8;
   change=999;
   iter=1;
   f=100.0;
   while ((iter<=3 || fabs(f)>eps) && iter<100)
   {
      if (iter==1){x1=T_guess; T=x1;}
      if (iter==2){x2=T_guess+0.1; T=x2;}
      if (iter>2) {T=x2;}
         f=h_m(Gas,Liq,T,p,xL)-h;
      if (iter==1){y1=f;}
      if (iter==2){y2=f;}
      if (iter>2)
      {
         y2=f;
         x3=x2-y2/(y2-y1)*(x2-x1);
         change=fabs(y2/(y2-y1)*(x2-x1));
         y1=y2; x1=x2; x2=x3;
      }
      iter=iter+1;
   }
   if (isNAN_FP(T))
      printf("uhoh");
   if (isINFINITY_FP(T))
      printf("uhoh");
   //printf("THP: %d %g %g %g %g\n",iter,f,T,p,h_m(Gas,Liq,T,p+20,xL));
   return T;
}
double T_Up(char *Gas, char *Liq, double U, double p, double xL, double V, double
    T_guess)
{
   double x1,x2,x3,y1,y2,eps,change,f=999.,T;
   int iter;
   eps=1e-8;
   change=999;
   iter=1;
   while ((iter<=3 || fabs(f)>eps) && iter<100)
   {
      if (iter==1){x1=T_guess; T=x1;}
      if (iter==2){x2=T_guess+0.1; T=x2;}
      if (iter>2) {T=x2;}
         f=U-u_m(Gas,Liq,T,p,xL)*rho_m(Gas,Liq,T,p,xL)*V;
      if (iter==1){y1=f;}
      if (iter>1)
      {
         y2=f;
         x3=x2-y2/(y2-y1)*(x2-x1);
         change=fabs(y2/(y2-y1)*(x2-x1));
         y1=y2; x1=x2; x2=x3;
      }
      iter=iter+1;
      if (iter>50)
      {
         printf("T_Up not converging with inputs\n U: %g p: %g xL: %g V: %g T_guess:
             %g\n",U,p,xL,V,T_guess);
      }
   }
   if (isNAN_FP(T))
      printf("uhoh");
   if (isINFINITY_FP(T))
      printf("uhoh");
   return T;
}
double p_Trho(char *Gas, char *Liq, double rho, double T, double xL, double p_guess)
{
   double x1,x2,x3,y1,y2,eps,change,f,p;
   int iter;
   eps=1e-8;
   change=999;
```

```
    iter=1;
    while ((iter<=3 || change>eps) && iter<100)
    {
        if (iter==1){x1=p_guess; p=x1;}
        if (iter==2){x2=p_guess+0.1; p=x2;}
        if (iter>2) {p=x2;}
            // Find the pressure which gives the same density
            f=rho_m(Gas,Liq,T,p,xL)-rho;
        if (iter==1){y1=f;}
        if (iter==2){y2=f;}
        if (iter>2)
        {
            y2=f;
            x3=x2-y2/(y2-y1)*(x2-x1);
            change=fabs(y2/(y2-y1)*(x2-x1));
            y1=y2; x1=x2; x2=x3;
        }
        iter=iter+1;
        if (iter>50)
        {
            printf("p_Trho not converging with inputs\n rho: %g T: %g xL: %g p_guess: %g
                \n",rho,T,xL,p_guess);
        }
    }
    if (isNAN_FP(x3))
        printf("uhoh");
    if (isINFINITY_FP(x3))
        printf("uhoh");
    return x3;
}
double h_sp(char *Gas, char *Liq, double s, double p, double xL, double T_guess)
{
    double x1,x2,x3,y1,y2,eps,change,f,T;
    int iter;
    eps=1e-8;
    change=999;
    iter=1;
    while ((iter<=3 || change>eps) && iter<100)
    {
        if (iter==1){x1=T_guess; T=x1;}
        if (iter==2){x2=T_guess+0.1; T=x2;}
        if (iter>2) {T=x2;}
            // Find the temperature which gives the same entropy
            f=s_m(Gas,Liq,T,p,xL)-s;
        if (iter==1){y1=f;}
        if (iter==2){y2=f;}
        if (iter>2)
        {
            y2=f;
            x3=x2-y2/(y2-y1)*(x2-x1);
            change=fabs(y2/(y2-y1)*(x2-x1));
            y1=y2; x1=x2; x2=x3;
        }
        iter=iter+1;
        if (iter>50)
        {
            printf("h_sp not converging with inputs\n s: %g p: %g xL: %gT_guess: %g\n",s
                ,p,xL,T_guess);
        }
    }
    if (isNAN_FP(h_m(Gas,Liq,T,p,xL)))
        printf("uhoh");
    if (isINFINITY_FP(h_m(Gas,Liq,T,p,xL)))
        printf("uhoh");
    //Evaluate the enthalpy at the constant-entropy temp
    return h_m(Gas,Liq,T,p,xL);
}
double T_sp(char *Gas, char *Liq, double s, double p, double xL, double T_guess)
{
    double x1,x2,x3,y1,y2,eps,change,f,T;
    int iter;
    eps=1e-8;
    change=999;
    iter=1;
    while ((iter<=3 || change>eps) && iter<100)
    {
        if (iter==1){x1=T_guess; T=x1;}
```

```
        if (iter==2){x2=T_guess+0.1; T=x2;}
        if (iter>2) {T=x2;}
            // Find the temperature which gives the same entropy
            f=s_m(Gas,Liq,T,p,xL)-s;
        if (iter==1){y1=f;}
        if (iter==2){y2=f;}
        if (iter>2)
        {
            y2=f;
            x3=x2-y2/(y2-y1)*(x2-x1);
            change=fabs(y2/(y2-y1)*(x2-x1));
            y1=y2; x1=x2; x2=x3;
        }
        iter=iter+1;
        if (iter>50)
        {
            printf("h_sp not converging with inputs\n s: %g p: %g xL: %gT_guess: %g\n",s
                ,p,xL,T_guess);
        }
    }
    if (isNAN_FP(h_m(Gas,Liq,T,p,xL)))
        printf("uhoh");
    if (isINFINITY_FP(h_m(Gas,Liq,T,p,xL)))
        printf("uhoh");
    //Return the temperature
    return T;
}
double dpdT_const_v(char *Gas, char *Liq, double T, double p1, double xL)
{
    double x1,x2,x3,y1,y2,eps,change,f,v1,delta,p2;
    int iter;
    delta=1e-5;
    eps=1e-6;
    change=999;
    iter=1;
    v1=1/rho_m(Gas,Liq,T,p1,xL);
    while ((iter<=3 || fabs(f)>eps) && iter<100)
    {
        if (iter==1){x1=p1; p2=x1;}
        if (iter==2){x2=p1+0.001; p2=x2;}
        if (iter>2) {p2=x2;}
            // Find the pressure which gives the same specific volume
            f=1.0/rho_m(Gas,Liq,T+delta,p2,xL)-v1;
        if (iter==1){y1=f;}
        if (iter>1)
        {
            y2=f;
            x3=x2-y2/(y2-y1)*(x2-x1);
            change=fabs(y2/(y2-y1)*(x2-x1));
            y1=y2; x1=x2; x2=x3;
        }
        iter=iter+1;
        if (iter>50)
        {
            printf("dpdT_const_v not converging with inputs\n T: %g p1: %g xL: %g\n",T,
                p1,xL);
        }
    }
    if (isNAN_FP((p2-p1)/delta))
        printf("uhoh");
    if (isINFINITY_FP((p2-p1)/delta))
        printf("uhoh");
    //Evaluate the enthalpy at the constant-entropy temp
    return (p2-p1)/delta;
}
//double c_g(char *Gas, double T, double p)
//{
// double dp_dv,k,v;
//
// v=1/rho_g(Gas,T,p);
// k=c_p(Gas,T,p)/c_v(Gas,T,p);
// dp_dv=(
//}
```

R744.h

```
void setCoeffs(void);
double p_R744(double T, double rho);
double rho_R744(double T, double p_rho, int Types);
double h_R744(double T, double p_rho, int Types);
double u_R744(double T, double p_rho, int Types);
double s_R744(double T, double p_rho, int Types);
double cv_R744(double T, double p_rho, int Types);
double cp_R744(double T, double p_rho, int Types);
double c_R744(double T, double p_rho, int Types);
double visc_R744(double T, double p_rho, int Types);
double k_R744(double T, double p_rho, int Types);
double w_R744(double T, double p_rho, int Types);
double MM_R744(void);
double rhosatV_R744(double T);
double rhosatL_R744(double T);
double pcrit_R744(void);
double Tcrit_R744(void);
double hsat_R744(double T,double x);
double rhosat_R744(double T,double x);
double ssat_R744(double T,double x);
double Psat_R744(double T);
double Tsat_R744(double P);
// Derivatives
double dhdrho_R744(double T, double p_rho, int Types);
double dhdT_R744(double T, double p_rho, int Types);
double dpdrho_R744(double T, double p_rho, int Types);
double dpdT_R744(double T, double p_rho, int Types);
int errCode_R744(void);
```

R744.c

```
/* Properties of Carbon Dioxide (R744)
by Ian Bell
Themo properties from
"A New Equation of State for Carbon Dioxide Covering the Fluid Region from the
Triple Point Temperature to 1100 K at Pressures up to 800 MPa",
R. Span and W. Wagner, J. Phys. Chem. Ref. Data, v. 25, 1996
WARNING: Thermal conductivity not coded!!
In order to call the exposed functions, rho_, h_, s_, cp_,..... there are three
different ways the inputs can be passed, and this is expressed by the Types integer
    flag.
These macros are defined in the PropMacros.h header file:
1) First parameter temperature, second parameter pressure ex: h_R410A(260,1785,1)
    =-67.53
   In this case, the lookup tables are built if needed and then interpolated
2) First parameter temperature, second parameter density ex: h_R410A(260,43.29,2)
    =-67.53
   Density and temp plugged directly into EOS
3) First parameter temperature, second parameter pressure ex: h_R410A(260,1785,3)
    =-67.53
   Density solved for, then plugged into EOS (can be quite slow)
*/
#if defined(_MSC_VER)
#define _CRTDBG_MAP_ALLOC
#define _CRT_SECURE_NO_WARNINGS
#include <stdlib.h>
#include <crtdbg.h>
#else
#include <stdlib.h>
#endif
#include "math.h"
#include "stdio.h"
#include <string.h>
#include "PropErrorCodes.h"
#include "PropMacros.h"
#include "R744.h"
#include "time.h"
static int errCode;
static char errStr[ERRSTRLENGTH];
#define nP 200
#define nT 200
```

```
const static double Tmin=220,Tmax=800, Pmin=500.03,Pmax=16000;
static double hmat[nT][nP];
static double rhomat[nT][nP];
static double cpmat[nT][nP];
static double smat[nT][nP];
static double cvmat[nT][nP];
static double cmat[nT][nP];
static double umat[nT][nP];
static double viscmat[nT][nP];
static double Tvec[nT];
static double pvec[nP];
static int TablesBuilt;
static double alpha[40],beta[43],GAMMA[40],epsilon[40],a[43],b[43],A[43],B[43],C[43],
    D[43],a0[9],theta0[9];
static const double Tc=304.128, R_R744=0.1889241, rhoc=467.6, Pc=7377.3;
                //    K              kJ/kg-K       kg/m^3          kPa
static const double n[]={0,
 0.3885682320316100E+00,
 0.2938547594274000E+01,
-0.5586718853493400E+01,
-0.7675319959247700E+00,
 0.3172900558041600E+00,
 0.5480331589776700E+00,
 0.1227941122033500E+00,
 0.2165896154322000E+01,
 0.1584173510972400E+01,
-0.2313270540550300E+00,
 0.5811691643143600E-01,
-0.5536913720538200E-00,
 0.4894661590942200E-00,
-0.2427573984350100E-01,
 0.6249479050167800E-01,
-0.1217586022524600E+00,
-0.3705568527008600E+00,
-0.1677587970042600E-01,
-0.1196073663798700E+00,
-0.4561936250877800E-01,
 0.3561278927034600E-01,
-0.7442772713205200E-02,
-0.1739570490243200E-02,
-0.2181012128952700E-01,
 0.2433216655923600E-01,
-0.3744013342346300E-01,
 0.1433871575687800E-00,
-0.1349196908328600E-00,
-0.2315122505348000E-01,
 0.1236312549290100E-01,
 0.2105832197294000E-02,
-0.3395851902636800E-03,
 0.5599365177159200E-02,
-0.3033511805564600E-03,
-0.2136548868832000E+03,
 0.2664156914927200E+05,
-0.2402721220455700E+05,
-0.2834160342399900E+03,
 0.2124728440017900E+03,
-0.6664227654075100E+00,
 0.7260863234989700E+00,
 0.5506866861284200E-01};
static const int d[]={0,
1,
1,
1,
1,
2,
2,
3,
1,
2,
4,
5,
5,
5,
6,
6,
6,
1,
1,
4,
4,
4,
7,
8,
2,
```

```
3,
3,
5,
5,
6,
7,
8,
10,
4,
8,
2,
2,
2,
3,
3};
static const double t[]={0.00,
0.00,
0.75,
1.00,
2.00,
0.75,
2.00,
0.75,
1.50,
1.50,
2.50,
0.00,
1.50,
2.00,
0.00,
1.00,
2.00,
3.00,
6.00,
3.00,
6.00,
8.00,
6.00,
0.00,
7.00,
12.00,
16.00,
22.00,
24.00,
16.00,
24.00,
8.00,
2.00,
28.00,
14.00,
1.00,
0.00,
1.00,
3.00,
3.00};
static const int c[]={0,0,0,0,0,0,0,0,
1,
1,
1,
1,
1,
1,
1,
1,
1,
2,
2,
2,
2,
2,
2,
2,
3,
3,
3,
4,
4,
4,
4,
4,
4,
5,
6};
void setCoeffs(void)
{
alpha[35]=25.0;
alpha[36]=25.0;
```

```
alpha[37]=25.0;
alpha[38]=15.0;
alpha[39]=20.0;

beta[35]=325.0;
beta[36]=300.0;
beta[37]=300.0;
beta[38]=275.0;
beta[39]=275.0;

GAMMA[35]=1.16;
GAMMA[36]=1.19;
GAMMA[37]=1.19;
GAMMA[38]=1.25;
GAMMA[39]=1.22;

epsilon[35]=1.00;
epsilon[36]=1.00;
epsilon[37]=1.00;
epsilon[38]=1.00;
epsilon[39]=1.00;

a[40]=3.5;
a[41]=3.5;
a[42]=3.0;

b[40]=0.875;
b[41]=0.925;
b[42]=0.875;

beta[40]=0.300;
beta[41]=0.300;
beta[42]=0.300;

A[40]=0.700;
A[41]=0.700;
A[42]=0.700;

B[40]=0.3;
B[41]=0.3;
B[42]=1.0;

C[40]=10.0;
C[41]=10.0;
C[42]=12.5;

D[40]=275.0;
D[41]=275.0;
D[42]=275.0;

//Constants for ideal gas expression
a0[1]=8.37304456;
a0[2]=-3.70454304;
a0[3]=2.500000;
a0[4]=1.99427042;
a0[5]=0.62105248;
a0[6]=0.41195293;
a0[7]=1.04028922;
a0[8]=0.08327678;

theta0[4]=3.15163;
theta0[5]=6.11190;
theta0[6]=6.77708;
theta0[7]=11.32384;
theta0[8]=27.08792;
}
static double powI(double x, int y);
static double QuadInterpolate(double x0, double x1, double x2, double f0, double f1,
    double f2, double x);

static double get_Delta(double T, double P);
static double Pressure_Trho(double T, double rho);
static double IntEnergy_Trho(double T, double rho);
static double Enthalpy_Trho(double T, double rho);
static double Entropy_Trho(double T, double rho);
static double SpecHeatV_Trho(double T, double rho);
static double SpecHeatP_Trho(double T, double rho);
static double SpeedSound_Trho(double T, double rho);

static double phir(double tau, double delta);
static double phi0(double tau, double delta);
static double dphir_dDelta(double tau, double delta);
static double dphir2_dDelta2(double tau, double delta);
static double dphir_dTau(double tau, double delta);
static double dphi0_dDelta(double tau, double delta);
static double dphi02_dDelta2(double tau, double delta);
static double dphi0_dTau(double tau, double delta);
static double dphi02_dTau2(double tau, double delta);
```

```
static double dphir2_dTau2(double tau, double delta);
static double dphir2_dDelta_dTau(double tau, double delta);
static double dhdT(double tau, double delta);
static double dhdrho(double tau, double delta);
static double dpdT(double tau, double delta);
static double dpdrho(double tau, double delta);
static double LookupValue(char *Prop,double T, double p);
static int isNAN(double x);
static int isINFINITY(double x);
static void WriteLookup(void)
{
   int i,j;
   FILE *fp_h,*fp_s,*fp_rho,*fp_u,*fp_cp,*fp_cv,*fp_visc;
   fp_h=fopen("h.csv","w");
   fp_s=fopen("s.csv","w");
   fp_u=fopen("u.csv","w");
   fp_cp=fopen("cp.csv","w");
   fp_cv=fopen("cv.csv","w");
   fp_rho=fopen("rho.csv","w");
   fp_visc=fopen("visc.csv","w");
   // Write the pressure header row
   for (j=0;j<nP;j++)
   {
      fprintf(fp_h,",%0.12f",pvec[j]);
      fprintf(fp_s,",%0.12f",pvec[j]);
      fprintf(fp_rho,",%0.12f",pvec[j]);
      fprintf(fp_u,",%0.12f",pvec[j]);
      fprintf(fp_cp,",%0.12f",pvec[j]);
      fprintf(fp_cv,",%0.12f",pvec[j]);
      fprintf(fp_visc,",%0.12f",pvec[j]);
   }
   fprintf(fp_h,"\n");
   fprintf(fp_s,"\n");
   fprintf(fp_rho,"\n");
   fprintf(fp_u,"\n");
   fprintf(fp_cp,"\n");
   fprintf(fp_cv,"\n");
   fprintf(fp_visc,"\n");
   for (i=1;i<nT;i++)
   {
      fprintf(fp_h,"%0.12f",Tvec[i]);
      fprintf(fp_s,"%0.12f",Tvec[i]);
      fprintf(fp_rho,"%0.12f",Tvec[i]);
      fprintf(fp_u,"%0.12f",Tvec[i]);
      fprintf(fp_cp,"%0.12f",Tvec[i]);
      fprintf(fp_cv,"%0.12f",Tvec[i]);
      fprintf(fp_visc,"%0.12f",Tvec[i]);
      for (j=0;j<nP;j++)
      {
         fprintf(fp_h,",%0.12f",hmat[i][j]);
         fprintf(fp_s,",%0.12f",smat[i][j]);
         fprintf(fp_rho,",%0.12f",rhomat[i][j]);
         fprintf(fp_u,",%0.12f",umat[i][j]);
         fprintf(fp_cp,",%0.12f",cpmat[i][j]);
         fprintf(fp_cv,",%0.12f",cvmat[i][j]);
         fprintf(fp_visc,",%0.12f",viscmat[i][j]);
      }
      fprintf(fp_h,"\n");
      fprintf(fp_s,"\n");
      fprintf(fp_rho,"\n");
      fprintf(fp_u,"\n");
      fprintf(fp_cp,"\n");
      fprintf(fp_cv,"\n");
      fprintf(fp_visc,"\n");
   }
   fclose(fp_h);
   fclose(fp_s);
   fclose(fp_rho);
   fclose(fp_u);
   fclose(fp_cp);
   fclose(fp_cv);
   fclose(fp_visc);
}
static void BuildLookup(void)
```

```
{
   int i,j;
   if (!TablesBuilt)
   {
      printf("Building Lookup Tables... Please wait...\n");
      for (i=0;i<nT;i++)
      {
         Tvec[i]=Tmin+i*(Tmax-Tmin)/(nT-1);
      }
      for (j=0;j<nP;j++)
      {
         pvec[j]=Pmin+j*(Pmax-Pmin)/(nP-1);
      }
      for (i=0;i<nT;i++)
      {
         for (j=0;j<nP;j++)
         {
            if (Tvec[i]>Tc || pvec[j]<Psat_R744(Tvec[i]))
            {
               rhomat[i][j]=get_Delta(Tvec[i],pvec[j])*rhoc;
               hmat[i][j]=h_R744(Tvec[i],rhomat[i][j],TYPE_Trho);
               smat[i][j]=s_R744(Tvec[i],rhomat[i][j],TYPE_Trho);
               umat[i][j]=u_R744(Tvec[i],rhomat[i][j],TYPE_Trho);
               cpmat[i][j]=cp_R744(Tvec[i],rhomat[i][j],TYPE_Trho);
               cvmat[i][j]=cv_R744(Tvec[i],rhomat[i][j],TYPE_Trho);
               cmat[i][j]=c_R744(Tvec[i],rhomat[i][j],TYPE_Trho);
               viscmat[i][j]=visc_R744(Tvec[i],rhomat[i][j],TYPE_Trho);
            }
            else
            {
               hmat[i][j]=_HUGE;
               smat[i][j]=_HUGE;
               umat[i][j]=_HUGE;
               rhomat[i][j]=_HUGE;
               umat[i][j]=_HUGE;
               cpmat[i][j]=_HUGE;
               cvmat[i][j]=_HUGE;
               cmat[i][j]=_HUGE;
               viscmat[i][j]=_HUGE;
            }
         }
      }
      TablesBuilt=1;
      //WriteLookup();
   }
}

/*********************************************/
/*          Public Property Functions        */
/*********************************************/
double rho_R744(double T, double p, int Types)
{
   setCoeffs();
   errCode=0; // Reset error code
   switch(Types)
   {
      case TYPE_TPNoLookup:
         return get_Delta(T,p)*rhoc;
      case TYPE_TP:
         BuildLookup();
         return LookupValue("rho",T,p);
      default:
         errCode=BAD_PROPCODE;
         return _HUGE;
   }
}
double p_R744(double T, double rho)
{
   setCoeffs();
   return Pressure_Trho(T,rho);
}
double h_R744(double T, double p_rho, int Types)
{
   double rho;
   setCoeffs();
   errCode=0; // Reset error code
   switch(Types)
   {
      case TYPE_Trho:
         return Enthalpy_Trho(T,p_rho);
```

```
        case TYPE_TPNoLookup:
            rho=get_Delta(T,p_rho)*rhoc;
            return Enthalpy_Trho(T,rho);
        case TYPE_TP:
            BuildLookup();
            return LookupValue("h",T,p_rho);
        default:
            errCode=BAD_PROPCODE;
            return _HUGE;
    }
}
double s_R744(double T, double p_rho, int Types)
{
    double rho;
    setCoeffs();
    errCode=0; // Reset error code
    switch(Types)
    {
        case TYPE_Trho:
            return Entropy_Trho(T,p_rho);
        case TYPE_TPNoLookup:
            rho=get_Delta(T,p_rho)*rhoc;
            return Entropy_Trho(T,rho);
        case TYPE_TP:
            BuildLookup();
            return LookupValue("s",T,p_rho);
        default:
            errCode=BAD_PROPCODE;
            return _HUGE;
    }
}
double u_R744(double T, double p_rho, int Types)
{
    double rho;
    setCoeffs();
    errCode=0; // Reset error code
    switch(Types)
    {
        case TYPE_Trho:
            return IntEnergy_Trho(T,p_rho);
        case TYPE_TPNoLookup:
            rho=get_Delta(T,p_rho)*rhoc;
            return IntEnergy_Trho(T,rho);
        case TYPE_TP:
            BuildLookup();
            return LookupValue("u",T,p_rho);
        default:
            errCode=BAD_PROPCODE;
            return _HUGE;
    }
}
double cp_R744(double T, double p_rho, int Types)
{
    double rho;
    setCoeffs();
    errCode=0; // Reset error code
    switch(Types)
    {
        case TYPE_Trho:
            return SpecHeatP_Trho(T,p_rho);
        case TYPE_TPNoLookup:
            rho=get_Delta(T,p_rho)*rhoc;
            return SpecHeatP_Trho(T,rho);
        case TYPE_TP:
            BuildLookup();
            return LookupValue("cp",T,p_rho);
        default:
            errCode=BAD_PROPCODE;
            return _HUGE;
    }
}
double cv_R744(double T, double p_rho, int Types)
{
    double rho;
    setCoeffs();
    errCode=0; // Reset error code
    switch(Types)
    {
        case TYPE_Trho:
            return SpecHeatV_Trho(T,p_rho);
        case TYPE_TPNoLookup:
            rho=get_Delta(T,p_rho)*rhoc;
```

```
            return SpecHeatV_Trho(T,rho);
        case TYPE_TP:
            BuildLookup();
            return LookupValue("cv",T,p_rho);
        default:
            errCode=BAD_PROPCODE;
            return _HUGE;
    }
}
double rhosatV_R744(double T)
{
    const double ti[]={0,0.340,1.0/2.0,1.0,7.0/3.0,14.0/3.0};
    const double ai[]={0,-1.7074879,-0.82274670,-4.6008549,-10.111178,-29.742252};
    double summer=0;
    int i;
    for (i=1;i<=5;i++)
    {
        summer=summer+ai[i]*pow(1.0-T/Tc,ti[i]);
    }
    return rhoc*exp(summer);
}
double rhosatL_R744(double T)
{
    const double ti[]={0,0.340,1.0/2.0,10.0/6.0,11.0/6.0};
    const double ai[]={0,1.9245108,-0.62385555,-0.32731127,0.39245142};
    double summer=0;
    int i;
    for (i=1;i<=4;i++)
    {
        summer=summer+ai[i]*pow(1.0-T/Tc,ti[i]);
    }
    return rhoc*exp(summer);
}
double w_R744(double T, double p_rho, int Types)
{
    double rho;
    errCode=0; // Reset error code
    switch(Types)
    {
        case TYPE_Trho:
            return SpeedSound_Trho(T,p_rho);
        case TYPE_TPNoLookup:
            rho=get_Delta(T,p_rho)*rhoc;
            return SpeedSound_Trho(T,rho);
        default:
            errCode=BAD_PROPCODE;
            return _HUGE;
    }
}
double c_R744(double T, double p_rho, int Types)
{
    double rho;
    if (Types==TYPE_Trho)
        return SpeedSound_Trho(T,p_rho);
    else
    {
        if (Types==TYPE_TP)
        {
            BuildLookup();
            return LookupValue("c",T,p_rho);
        }
        else //Types==TYPE_TPNoTable
        {
            rho=get_Delta(T,p_rho)*rhoc;
            return SpeedSound_Trho(T,rho);
        }
    }
}
double visc_R744(double T,double p_rho, int Types)
{
    int i;
    double e_k=251.196,Tstar,sumGstar=0.0,Gstar,eta0,delta_eta,rho;
    double a[]={0.235156,-0.491266,5.211155e-2,5.347906e-2,-1.537102e-2};
    double d11=0.4071119e-2,d21=0.7198037e-4,d64=0.2411697e-16,d81=0.2971072e-22,d82
        =-0.1627888e-22;
    if (Types==TYPE_Trho)
        rho=p_rho;
    else if (Types==TYPE_TPNoLookup)
    {
        rho=get_Delta(T,p_rho)*rhoc;
```

```
    }
    else if (Types==TYPE_TP)
    {
        BuildLookup();
        return LookupValue("visc",T,p_rho);
    }
    Tstar=T/e_k;
    for (i=0;i<=4;i++)
    {
        sumGstar=sumGstar+a[i]*powI(log(Tstar),i);
    }
    Gstar=exp(sumGstar);
    eta0=1.00697*sqrt(T)/Gstar;
    delta_eta=d11*rho+d21*rho*rho*d64*powI(rho,6)/powI(Tstar,3)+d81*powI(rho,8)+d82*
        powI(rho,8)/Tstar;
    return (eta0+delta_eta)/1e6;
}
double k_R744(double T,double p_rho, int Types)
{
    fprintf(stderr,"Thermal conductivity not coded for R744 (CO2).  Sorry.\n");
    return _HUGE;
}
double MM_R744(void)
{
    return 44.01;
}
double pcrit_R744(void)
{
    return Pc;
}
double Tcrit_R744(void)
{
    return Tc;
}
int errCode_R744(void)
{
    return errCode;
}
double dhdT_R744(double T, double p_rho, int Types)
{
    double rho;
    errCode=0; // Reset error code
    switch(Types)
    {
        case TYPE_Trho:
            rho=p_rho;
            return dhdT(Tc/T,rho/rhoc);
        case TYPE_TPNoLookup:
            rho=get_Delta(T,p_rho)*rhoc;
            return dhdT(Tc/T,rho/rhoc);
        case 99:
            rho=get_Delta(T,p_rho)*rhoc;
            return (Enthalpy_Trho(T+0.001,rho)-Enthalpy_Trho(T,rho))/0.001;
        default:
            errCode=BAD_PROPCODE;
            return _HUGE;
    }
}
double dhdrho_R744(double T, double p_rho, int Types)
{
    double rho;
    errCode=0; // Reset error code
    switch(Types)
    {
        case TYPE_Trho:
            rho=p_rho;
            return dhdrho(Tc/T,rho/rhoc);
        case TYPE_TPNoLookup:
            rho=get_Delta(T,p_rho)*rhoc;
            return dhdrho(Tc/T,rho/rhoc);
        case 99:
            rho=get_Delta(T,p_rho)*rhoc;
            return (Enthalpy_Trho(T,rho+0.001)-Enthalpy_Trho(T,rho))/0.001;
        default:
            errCode=BAD_PROPCODE;
            return _HUGE;
    }
}
```

```
double dpdT_R744(double T, double p_rho, int Types)
{
   double rho;
   errCode=0; // Reset error code
   switch(Types)
   {
      case TYPE_Trho:
         rho=p_rho;
         return dpdT(Tc/T,rho/rhoc);
      case TYPE_TPNoLookup:
         rho=get_Delta(T,p_rho)*rhoc;
         return dpdT(Tc/T,rho/rhoc);
      case 99:
         rho=get_Delta(T,p_rho)*rhoc;
         return (Pressure_Trho(T+0.01,rho)-Pressure_Trho(T,rho))/0.01;
      default:
         errCode=BAD_PROPCODE;
         return _HUGE;
   }
}
double dpdrho_R744(double T, double p_rho, int Types)
{
   double rho;
   errCode=0; // Reset error code
   switch(Types)
   {
      case TYPE_Trho:
         rho=p_rho;
         return dpdrho(Tc/T,rho/rhoc);
      case TYPE_TPNoLookup:
         rho=get_Delta(T,p_rho)*rhoc;
         return dpdrho(Tc/T,rho/rhoc);
      case 99:
         rho=get_Delta(T,p_rho)*rhoc;
         return (Pressure_Trho(T,rho+0.0001)-Pressure_Trho(T,rho))/0.0001;
      default:
         errCode=BAD_PROPCODE;
         return _HUGE;
   }
}
/***********************************************/
/*          Private Property Functions         */
/***********************************************/
static double Pressure_Trho(double T, double rho)
{
   double delta,tau;
   delta=rho/rhoc;
   tau=Tc/T;
   return R_R744*T*rho*(1.0+delta*dphir_dDelta(tau,delta));
}
static double IntEnergy_Trho(double T, double rho)
{
   double delta,tau;
   delta=rho/rhoc;
   tau=Tc/T;
   return R_R744*T*tau*(dphi0_dTau(tau,delta)+dphir_dTau(tau,delta));
}
static double Enthalpy_Trho(double T, double rho)
{
   double delta,tau;
   delta=rho/rhoc;
   tau=Tc/T;
   return R_R744*T*(1+tau*(dphi0_dTau(tau,delta)+dphir_dTau(tau,delta))+delta*
       dphir_dDelta(tau,delta));
}
static double Entropy_Trho(double T, double rho)
{
   double delta,tau;
   delta=rho/rhoc;
   tau=Tc/T;
   return R_R744*(tau*(dphi0_dTau(tau,delta)+dphir_dTau(tau,delta))-phi0(tau,delta)-
       phir(tau,delta));
}
static double SpecHeatV_Trho(double T, double rho)
{
   double delta,tau;
   delta=rho/rhoc;
   tau=Tc/T;
   return -R_R744*powI(tau,2)*(dphi02_dTau2(tau,delta)+dphir2_dTau2(tau,delta));
}
```

```
static double SpecHeatP_Trho(double T, double rho)
{
   double delta,tau,c1,c2;
   delta=rho/rhoc;
   tau=Tc/T;
   c1=powI(1.0+delta*dphir_dDelta(tau,delta)-delta*tau*dphir2_dDelta_dTau(tau,delta)
       ,2);
    c2=(1.0+2.0*delta*dphir_dDelta(tau,delta)+powI(delta,2)*dphir2_dDelta2(tau,delta)
        );
    return R_R744*(-powI(tau,2)*(dphi02_dTau2(tau,delta)+dphir2_dTau2(tau,delta))+c1/
        c2);
}
static double SpeedSound_Trho(double T, double rho)
{
   double delta,tau,c1,c2;
   delta=rho/rhoc;
   tau=Tc/T;
   c1=-SpecHeatV_Trho(T,rho)/R_R744;
   c2=(1.0+2.0*delta*dphir_dDelta(tau,delta)+powI(delta,2)*dphir2_dDelta2(tau,delta))
       ;
    return sqrt(-c2*T*SpecHeatP_Trho(T,rho)*1000/c1);
}
/***********************************************/
/*          Property Derivatives               */
/***********************************************/
// See Lemmon, 2000 for more information
static double dhdrho(double tau, double delta)
{
   double T,R;
   T=Tc/tau;   R=R_R744;
   //Note: dphi02_dDelta_dTau(tau,delta) is equal to zero
   return R*T/rhoc*(tau*(dphir2_dDelta_dTau(tau,delta))+dphir_dDelta(tau,delta)+delta
       *dphir2_dDelta2(tau,delta));
}
static double dhdT(double tau, double delta)
{
   double dhdT_rho,T,R,dhdtau;
   T=Tc/tau;   R=R_R744;
   dhdT_rho=R*tau*(dphi0_dTau(tau,delta)+dphir_dTau(tau,delta))+R*delta*dphir_dDelta(
       tau,delta)+R;
   dhdtau=R*T*(dphi0_dTau(tau,delta)+ dphir_dTau(tau,delta))+R*T*tau*(dphi02_dTau2(
       tau,delta)+dphir2_dTau2(tau,delta))+R*T*delta*dphir2_dDelta_dTau(tau,delta);
   return dhdT_rho+dhdtau*(-Tc/T/T);
}
static double dpdT(double tau, double delta)
{
   double T,R,rho;
   T=Tc/tau;   R=R_R744;  rho=delta*rhoc;
   return rho*R*(1+delta*dphir_dDelta(tau,delta)-delta*tau*dphir2_dDelta_dTau(tau,
       delta));
}
static double dpdrho(double tau, double delta)
{
   double T,R,rho;
   T=Tc/tau;   R=R_R744;  rho=delta*rhoc;
   return R*T*(1+2*delta*dphir_dDelta(tau,delta)+delta*delta*dphir2_dDelta2(tau,delta
       ));
}
/***********************************************/
/*          Private Property Functions         */
/***********************************************/
static double phir(double tau, double delta)
{
    int i;
    double phir=0,theta,DELTA,PSI,psi;
    for (i=1;i<=7;i++)
    {
        phir=phir+n[i]*powI(delta,d[i])*pow(tau,t[i]);
    }
    for (i=8;i<=34;i++)
    {
        phir=phir+n[i]*powI(delta,d[i])*pow(tau,t[i])*exp(-powI(delta,c[i]));
    }
    for (i=35;i<=39;i++)
    {
        psi=exp(-alpha[i]*powI(delta-epsilon[i],2)-beta[i]*powI(tau-GAMMA[i],2));
        phir=phir+n[i]*powI(delta,d[i])*pow(tau,t[i])*psi;
```

```
    }
    for (i=40;i<=42;i++)
    {
        theta=(1.0-tau)+A[i]*pow(powI(delta-1.0,2),1/(2*beta[i]));
        DELTA=powI(theta,2)+B[i]*pow(powI(delta-1.0,2),a[i]);
        PSI=exp(-C[i]*powI(delta-1.0,2)-D[i]*powI(tau-1.0,2));
        phir=phir+n[i]*pow(DELTA,b[i])*delta*PSI;
    }
    return phir;
}

static double dphir_dDelta(double tau, double delta)
{
    int i;
    double dphir_dDelta=0,theta,DELTA,PSI,dPSI_dDelta,dDELTA_dDelta,dDELTAbi_dDelta,
        psi;
    double di, ci;
    for (i=1;i<=7;i++)
    {
        di=(double)d[i];
        dphir_dDelta=dphir_dDelta+n[i]*di*powI(delta,d[i]-1)*pow(tau,t[i]);
    }
    for (i=8;i<=34;i++)
    {
        di=(double)d[i];
        ci=(double)c[i];
        dphir_dDelta=dphir_dDelta+n[i]*exp(-powI(delta,c[i]))*(powI(delta,d[i]-1)*pow
            (tau,t[i])*(di-ci*powI(delta,c[i])));
    }
    for (i=35;i<=39;i++)
    {
        di=(double)d[i];
        psi=exp(-alpha[i]*powI(delta-epsilon[i],2)-beta[i]*powI(tau-GAMMA[i],2));
        dphir_dDelta=dphir_dDelta+n[i]*powI(delta,d[i])*pow(tau,t[i])*psi*(di/delta
            -2.0*alpha[i]*(delta-epsilon[i]));
    }
    for (i=40;i<=42;i++)
    {
        theta=(1.0-tau)+A[i]*pow(powI(delta-1.0,2),1.0/(2.0*beta[i]));
        DELTA=powI(theta,2)+B[i]*pow(powI(delta-1.0,2),a[i]);
        PSI=exp(-C[i]*powI(delta-1.0,2)-D[i]*powI(tau-1.0,2));
        dPSI_dDelta=-2.0*C[i]*(delta-1.0)*PSI;
        dDELTA_dDelta=(delta-1.0)*(A[i]*theta*2.0/beta[i]*pow(powI(delta-1.0,2)
            ,1.0/(2.0*beta[i])-1.0)+2.0*B[i]*a[i]*pow(powI(delta-1.0,2),a[i]-1.0));
        dDELTAbi_dDelta=b[i]*pow(DELTA,b[i]-1.0)*dDELTA_dDelta;
        dphir_dDelta=dphir_dDelta+n[i]*(pow(DELTA,b[i])*(PSI+delta*dPSI_dDelta)+
            dDELTAbi_dDelta*delta*PSI);
    }
    return dphir_dDelta;
}
static double dphir2_dDelta2(double tau, double delta)
{
    int i;
    double di,ci;
    double dphir2_dDelta2=0,theta,DELTA,PSI,dPSI_dDelta,dDELTA_dDelta,dDELTAbi_dDelta
        ,psi,dPSI2_dDelta2,dDELTAbi2_dDelta2,dDELTA2_dDelta2;
    for (i=1;i<=7;i++)
    {
        di=(double)d[i];
        dphir2_dDelta2=dphir2_dDelta2+n[i]*di*(di-1.0)*powI(delta,d[i]-2)*pow(tau,t[i
            ]);
    }
    for (i=8;i<=34;i++)
    {
        di=(double)d[i];
        ci=(double)c[i];
        dphir2_dDelta2=dphir2_dDelta2+n[i]*exp(-powI(delta,c[i]))*(powI(delta,d[i]-2)
            *pow(tau,t[i])*( (di-ci*powI(delta,c[i]))*(di-1.0-ci*powI(delta,c[i])) -
            ci*ci*powI(delta,c[i])));
    }
    for (i=35;i<=39;i++)
    {
        di=(double)d[i];
        psi=exp(-alpha[i]*powI(delta-epsilon[i],2)-beta[i]*powI(tau-GAMMA[i],2));
        dphir2_dDelta2=dphir2_dDelta2+n[i]*pow(tau,t[i])*psi*(-2.0*alpha[i]*powI(
            delta,d[i])+4.0*powI(alpha[i],2)*powI(delta,d[i])*powI(delta-epsilon[i
            ],2)-4.0*di*alpha[i]*powI(delta,d[i]-1)*(delta-epsilon[i])+di*(di-1.0)*
            powI(delta,d[i]-2));
```

```
    }
    for (i=40;i<=42;i++)
    {
        theta=(1.0-tau)+A[i]*pow(powI(delta-1.0,2),1.0/(2.0*beta[i]));
        DELTA=powI(theta,2)+B[i]*pow(powI(delta-1.0,2),a[i]);
        PSI=exp(-C[i]*powI(delta-1.0,2)-D[i]*powI(tau-1.0,2));
        dPSI_dDelta=-2.0*C[i]*(delta-1.0)*PSI;
        dDELTA_dDelta=(delta-1.0)*(A[i]*theta*2.0/beta[i]*pow(powI(delta-1.0,2)
            ,1.0/(2.0*beta[i])-1.0)+2.0*B[i]*a[i]*pow(powI(delta-1.0,2),a[i]-1.0));
        dDELTAbi_dDelta=b[i]*pow(DELTA,b[i]-1.0)*dDELTA_dDelta;
        dPSI2_dDelta2=(2.0*C[i]*powI(delta-1.0,2)-1.0)*2.0*C[i]*PSI;
        dDELTA2_dDelta2=1.0/(delta-1.0)*dDELTA_dDelta+powI(delta-1.0,2)*(4.0*B[i]*a[i
            ]*(a[i]-1.0)*pow(powI(delta-1.0,2),a[i]-2.0)+2.0*powI(A[i]/beta[i],2)*
            powI(pow(powI(delta-1.0,2),1.0/(2.0*beta[i])-1.0),2)+A[i]*theta*4.0/beta[
            i]*(1.0/(2.0*beta[i])-1.0)*pow(powI(delta-1.0,2),1.0/(2.0*beta[i])-2.0));
        dDELTAbi2_dDelta2=b[i]*(pow(DELTA,b[i]-1.0)*dDELTA2_dDelta2+(b[i]-1.0)*pow(
            DELTA,b[i]-2.0)*powI(dDELTA_dDelta,2));
        dphir2_dDelta2=dphir2_dDelta2+n[i]*(pow(DELTA,b[i])*(2.0*dPSI_dDelta+delta*
            dPSI2_dDelta2)+2.0*dDELTAbi_dDelta*(PSI+delta*dPSI_dDelta)+
            dDELTAbi2_dDelta2*delta*PSI);
    }
    return dphir2_dDelta2;
}
static double dphir2_dDelta_dTau(double tau, double delta)
{
    int i;
    double di, ci;
    double dphir2_dDelta_dTau=0,theta,DELTA,PSI,dPSI_dDelta,dDELTA_dDelta,
        dDELTAbi_dDelta,psi,dPSI2_dDelta2,dDELTAbi2_dDelta2,dDELTA2_dDelta2;
    double dPSI2_dDelta_dTau, dDELTAbi2_dDelta_dTau, dPSI_dTau, dDELTAbi_dTau,
        dPSI2_dTau2, dDELTAbi2_dTau2;
    for (i=1;i<=7;i++)
    {
        di=(double)d[i];
        dphir2_dDelta_dTau=dphir2_dDelta_dTau+n[i]*di*t[i]*powI(delta,d[i]-1)*pow(tau
            ,t[i]-1.0);
    }
    for (i=8;i<=34;i++)
    {
        di=(double)d[i];
        ci=(double)c[i];
        dphir2_dDelta_dTau=dphir2_dDelta_dTau+n[i]*exp(-powI(delta,c[i]))*powI(delta,
            d[i]-1)*t[i]*pow(tau,t[i]-1.0)*(di-ci*powI(delta,c[i]));
    }
    for (i=35;i<=39;i++)
    {
        di=(double)d[i];
        psi=exp(-alpha[i]*powI(delta-epsilon[i],2)-beta[i]*powI(tau-GAMMA[i],2));
        dphir2_dDelta_dTau=dphir2_dDelta_dTau+n[i]*powI(delta,d[i])*pow(tau,t[i])*psi
            *(di/delta-2.0*alpha[i]*(delta-epsilon[i]))*(t[i]/tau-2.0*beta[i]*(tau-
            GAMMA[i]));
    }
    for (i=40;i<=42;i++)
    {
        theta=(1.0-tau)+A[i]*pow(powI(delta-1.0,2),1.0/(2.0*beta[i]));
        DELTA=powI(theta,2)+B[i]*pow(powI(delta-1.0,2),a[i]);
        PSI=exp(-C[i]*powI(delta-1.0,2)-D[i]*powI(tau-1.0,2));
        dPSI_dDelta=-2.0*C[i]*(delta-1.0)*PSI;
        dDELTA_dDelta=(delta-1.0)*(A[i]*theta*2.0/beta[i]*pow(powI(delta-1.0,2)
            ,1.0/(2.0*beta[i])-1.0)+2.0*B[i]*a[i]*pow(powI(delta-1.0,2),a[i]-1.0));
        dDELTAbi_dDelta=b[i]*pow(DELTA,b[i]-1.0)*dDELTA_dDelta;
        dPSI2_dDelta2=(2.0*C[i]*powI(delta-1.0,2)-1.0)*2.0*C[i]*PSI;
        dDELTA2_dDelta2=1.0/(delta-1.0)*dDELTA_dDelta+powI(delta-1.0,2)*(4.0*B[i]*a[i
            ]*(a[i]-1.0)*pow(powI(delta-1.0,2),a[i]-2.0)+2.0*powI(A[i]/beta[i],2)*
            powI(pow(powI(delta-1.0,2),1.0/(2.0*beta[i])-1.0),2)+A[i]*theta*4.0/beta[
            i]*(1.0/(2.0*beta[i])-1.0)*pow(powI(delta-1.0,2),1.0/(2.0*beta[i])-2.0));
        dDELTAbi2_dDelta2=b[i]*(pow(DELTA,b[i]-1.0)*dDELTA2_dDelta2+(b[i]-1.0)*pow(
            DELTA,b[i]-2.0)*powI(dDELTA_dDelta,2));
        dPSI_dTau=-2.0*D[i]*(tau-1.0)*PSI;
        dDELTAbi_dTau=-2.0*theta*b[i]*pow(DELTA,b[i]-1.0);
        dPSI2_dTau2=(2.0*D[i]*powI(tau-1.0,2)-1.0)*2.0*D[i]*PSI;
        dDELTAbi2_dTau2=2.0*b[i]*pow(DELTA,b[i]-1.0)+4.0*powI(theta,2)*b[i]*(b[i
            ]-1.0)*pow(DELTA,b[i]-2.0);
        dPSI2_dDelta_dTau=4.0*C[i]*D[i]*(delta-1.0)*(tau-1.0)*PSI;
```

```
        dDELTAbi2_dDelta_dTau=-A[i]*b[i]*2.0/beta[i]*pow(DELTA,b[i]-1.0)*(delta-1.0)*
            pow(powI(delta-1.0,2),1.0/(2.0*beta[i])-1.0)-2.0*theta*b[i]*(b[i]-1.0)*
            pow(DELTA,b[i]-2.0)*dDELTA_dDelta;
        dphir2_dDelta_dTau=dphir2_dDelta_dTau+n[i]*(pow(DELTA,b[i])*(dPSI_dTau+delta*
            dPSI2_dDelta_dTau)+delta*dDELTAbi_dDelta*dPSI_dTau+ dDELTAbi_dTau*(PSI+
            delta*dPSI_dDelta)+dDELTAbi2_dDelta_dTau*delta*PSI);
    }
    return dphir2_dDelta_dTau;
}
static double dphir_dTau(double tau, double delta)
{
    int i;
    double dphir_dTau=0,theta,DELTA,PSI,dPSI_dTau,dDELTAbi_dTau,psi;
    for (i=1;i<=7;i++)
    {
        dphir_dTau=dphir_dTau+n[i]*t[i]*powI(delta,d[i])*pow(tau,t[i]-1.0);
    }
    for (i=8;i<=34;i++)
    {
        dphir_dTau=dphir_dTau+n[i]*t[i]*powI(delta,d[i])*pow(tau,t[i]-1.0)*exp(-powI(
            delta,c[i]));
    }
    for (i=35;i<=39;i++)
    {
        psi=exp(-alpha[i]*powI(delta-epsilon[i],2)-beta[i]*powI(tau-GAMMA[i],2));
        dphir_dTau=dphir_dTau+n[i]*powI(delta,d[i])*pow(tau,t[i])*psi*(t[i]/tau-2.0*
            beta[i]*(tau-GAMMA[i]));
    }
    for (i=40;i<=42;i++)
    {
        theta=(1.0-tau)+A[i]*pow(powI(delta-1.0,2),1.0/(2.0*beta[i]));
        DELTA=powI(theta,2)+B[i]*pow(powI(delta-1.0,2),a[i]);
        PSI=exp(-C[i]*powI(delta-1.0,2)-D[i]*powI(tau-1.0,2));
        dPSI_dTau=-2.0*D[i]*(tau-1.0)*PSI;
        dDELTAbi_dTau=-2.0*theta*b[i]*pow(DELTA,b[i]-1.0);
        dphir_dTau=dphir_dTau+n[i]*delta*(dDELTAbi_dTau*PSI+pow(DELTA,b[i])*dPSI_dTau
            );
    }
    return dphir_dTau;
}
static double dphir2_dTau2(double tau, double delta)
{
    int i;
    double dphir2_dTau2=0,theta,DELTA,PSI,dPSI_dTau,dDELTAbi_dTau,psi,dPSI2_dTau2,
        dDELTAbi2_dTau2;
    for (i=1;i<=7;i++)
    {
        dphir2_dTau2=dphir2_dTau2+n[i]*t[i]*(t[i]-1.0)*powI(delta,d[i])*pow(tau,t[i
            ]-2.0);
    }
    for (i=8;i<=34;i++)
    {
        dphir2_dTau2=dphir2_dTau2+n[i]*t[i]*(t[i]-1.0)*powI(delta,d[i])*pow(tau,t[i
            ]-2.0)*exp(-powI(delta,c[i]));
    }
    for (i=35;i<=39;i++)
    {
        psi=exp(-alpha[i]*powI(delta-epsilon[i],2)-beta[i]*powI(tau-GAMMA[i],2));
        dphir2_dTau2=dphir2_dTau2+n[i]*powI(delta,d[i])*pow(tau,t[i])*psi*(powI(t[i]/
            tau-2.0*beta[i]*(tau-GAMMA[i]),2)-t[i]/powI(tau,2)-2.0*beta[i]);
    }
    for (i=40;i<=42;i++)
    {
        theta=(1.0-tau)+A[i]*pow(powI(delta-1.0,2),1/(2*beta[i]));
        DELTA=powI(theta,2)+B[i]*pow(powI(delta-1.0,2),a[i]);
        PSI=exp(-C[i]*powI(delta-1.0,2)-D[i]*powI(tau-1.0,2));
        dPSI_dTau=-2.0*D[i]*(tau-1.0)*PSI;
        dDELTAbi_dTau=-2.0*theta*b[i]*pow(DELTA,b[i]-1.0);
        dPSI2_dTau2=(2.0*D[i]*powI(tau-1.0,2)-1.0)*2.0*D[i]*PSI;
        dDELTAbi2_dTau2=2.0*b[i]*pow(DELTA,b[i]-1.0)+4.0*powI(theta,2)*b[i]*(b[i
            ]-1.0)*pow(DELTA,b[i]-2.0);
        dphir2_dTau2=dphir2_dTau2+n[i]*delta*(dDELTAbi2_dTau2*PSI+2.0*dDELTAbi_dTau*
            dPSI_dTau+pow(DELTA,b[i])*dPSI2_dTau2);
    }
```

```
    return dphir2_dTau2;
}
static double dphi0_dDelta(double tau, double delta)
{
    return 1/delta;
}
static double dphi02_dDelta2(double tau, double delta)
{
    return -1.0/powI(delta,2);
}
static double phi0(double tau, double delta)
{
    double phi0=0;
    int i;
    phi0=log(delta)+a0[1]+a0[2]*tau+a0[3]*log(tau);
    for (i=4;i<=8;i++)
    {
        phi0=phi0+a0[i]*log(1.0-exp(-theta0[i]*tau));
    }
    return phi0;
}
static double dphi0_dTau(double tau, double delta)
{
    double dphi0_dTau=0;
    int i;
    dphi0_dTau=a0[2]+a0[3]/tau;
    for (i=4;i<=8;i++)
    {
        dphi0_dTau=dphi0_dTau+a0[i]*theta0[i]*(1.0/(1.0-exp(-theta0[i]*tau))-1.0);
    }
    return dphi0_dTau;
}
static double dphi02_dTau2(double tau, double delta)
{
    double dphi02_dTau2=0;
    int i;
    dphi02_dTau2=-a0[3]/powI(tau,2);
    for (i=4;i<=8;i++)
    {
        dphi02_dTau2=dphi02_dTau2-a0[i]*powI(theta0[i],2)*exp(-theta0[i]*tau)/powI
            (1.0-exp(-theta0[i]*tau),2);
    }
    return dphi02_dTau2;
}
static double get_Delta(double T, double P)
{
    double change,eps=.0005;
    int counter=1;
    double r1,r2,r3,delta1,delta2,delta3;
    double tau;
    double delta_guess;
     setCoeffs();
    if (P>Pc)
    {
        if (T>Tc)
        {
            delta_guess=P/(R_R744*T)/rhoc;
        }
        else
        {
            delta_guess=1000/rhoc;
        }
    }
    else
    {
        if (T>Tsat_R744(P))
        {
            delta_guess=P/(R_R744*T)/rhoc;
        }
        else
        {
            delta_guess=1000/rhoc;
        }
    }
    tau=Tc/T;
    delta1=delta_guess;
    delta2=delta_guess+.00001;
    r1=P/(delta1*rhoc*R_R744*T)-1.0-delta1*dphir_dDelta(tau,delta1);
```

```
    r2=P/(delta2*rhoc*R_R744*T)-1.0-delta2*dphir_dDelta(tau,delta2);
    // End at change less than 0.05%
    while(counter==1 || (fabs(change)/fabs(delta2)>eps && counter<40))
    {
        delta3=delta2-r2/(r2-r1)*(delta2-delta1);
        r3=P/(delta3*rhoc*R_R744*T)-1.0-delta3*dphir_dDelta(tau,delta3);
        change=r2/(r2-r1)*(delta2-delta1);
        delta1=delta2;
        delta2=delta3;
        r1=r2;
        r2=r3;
        counter=counter+1;
//          mexPrintf("Iteration: %i \n",counter);
//          mexPrintf("%g \t %g \t %g \t %g \t %g\n",delta2,r2,change,T,P);
    }
   //printf("Iteration: %d \n",counter);
    return delta3;
}
double Psat_R744(double T)
{
    const double ti[]={0,1.0,1.5,2.0,4.0};
    const double ai[]={0,-7.0602087,1.9391218,-1.6463597,-3.2995634};
    double summer=0;
    int i;
    setCoeffs();
    for (i=1;i<=4;i++)
    {
        summer=summer+ai[i]*pow(1-T/Tc,ti[i]);
    }
    return Pc*exp(Tc/T*summer);
}
double Tsat_R744(double P)
{
    double change,eps=.00005;
    int counter=1;
    double r1,r2,r3,T1,T2,T3;
    T1=275;
    T2=275+.01;
    r1=Psat_R744(T1)-P;
    r2=Psat_R744(T2)-P;
    // End at change less than 0.5%
    while(counter==1 || (fabs(change)/fabs(T2)>eps && counter<40))
    {
        T3=T2-0.5*r2/(r2-r1)*(T2-T1);
        r3=Psat_R744(T3)-P;
        change=0.5*r2/(r2-r1)*(T2-T1);
        T1=T2;
        T2=T3;
        r1=r2;
        r2=r3;
        counter=counter+1;
    }
    return T3;
}
double hsat_R744(double T, double x)
{
    double delta,tau;
    if (x>0.5)
    {
        delta=rhosatV_R744(T)/rhoc;
        tau=Tc/T;
        return R_R744*T*(1+tau*(dphi0_dTau(tau,delta)+dphir_dTau(tau,delta))+delta*
            dphir_dDelta(tau,delta));
    }
    else
    {
        delta=rhosatL_R744(T)/rhoc;
        tau=Tc/T;
        return R_R744*T*(1+tau*(dphi0_dTau(tau,delta)+dphir_dTau(tau,delta))+delta*
            dphir_dDelta(tau,delta));
    }
}
double ssat_R744(double T, double x)
{
    double delta,tau;
    if (x>0.5)
    {
        delta=rhosatV_R744(T)/rhoc;
        tau=Tc/T;
```

```
        return R_R744*(tau*(dphi0_dTau(tau,delta)+dphir_dTau(tau,delta))-phi0(tau,
            delta)-phir(tau,delta));
    }
    else
    {
        delta=rhosatL_R744(T)/rhoc;
        tau=Tc/T;
        return R_R744*(tau*(dphi0_dTau(tau,delta)+dphir_dTau(tau,delta))-phi0(tau,
            delta)-phir(tau,delta));
    }
}
double rhosat_R744(double T, double x)
{
    if (x>0.5)
    {
        return rhosatV_R744(T);
    }
    else
    {
        return rhosatL_R744(T);
    }
}
static double LookupValue(char *Prop, double T, double p)
{
   int iPlow, iPhigh, iTlow, iThigh,L,R,M,iter;
   double T1, T2, T3, P1, P2, P3, y1, y2, y3, a1, a2, a3;
   double (*mat)[nT][nP];
   //Input checking
   if (T>Tmax || T<Tmin)
   {
      errCode=OUT_RANGE_T;
   }
   if (p>Pmax || p<Pmin)
   {
      errCode=OUT_RANGE_P;
   }
   if (T<Tmin || T > Tmax || p<Pmin || p>Pmax || isNAN(T) || isINFINITY(T) ||isNAN(p)
     || isINFINITY(p))
    {
      printf("Inputs to LookupValue(%s) of R744 out of bounds:  T=%g K \t P=%g kPa \n
          ",Prop,T,p);
    }
   L=0;
   R=nT-1;
   M=(L+R)/2;
   iter=0;
   // Use interval halving to find the indices which bracket the temperature of
       interest
   while (R-L>1)
   {
      if (T>=Tvec[M])
      { L=M; M=(L+R)/2; continue;}
      if (T<Tvec[M])
      { R=M; M=(L+R)/2; continue;}
      iter++;
      if (iter>100)
         printf("Problem with T(%g K)\n",T);
   }
   iTlow=L; iThigh=R;
   L=0;
   R=nP-1;
   M=(L+R)/2;
   iter=0;
   // Use interval halving to find the indices which bracket the pressure of interest
   while (R-L>1)
   {
      if (p>=pvec[M])
      { L=M; M=(L+R)/2; continue;}
      if (p<pvec[M])
      { R=M; M=(L+R)/2; continue;}
      iter++;
      if (iter>100)
         printf("Problem with p(%g kPa)\n",p);
   }
   iPlow=L; iPhigh=R;
   /* Depending on which property is desired,
   make the matrix mat a pointer to the
   desired property matrix */
   if (!strcmp(Prop,"rho"))
      mat=&rhomat;
```

```
    else if (!strcmp(Prop,"cp"))
        mat=&cpmat;
    else if (!strcmp(Prop,"cv"))
        mat=&cvmat;
    else if (!strcmp(Prop,"h"))
        mat=&hmat;
    else if (!strcmp(Prop,"s"))
        mat=&smat;
    else if (!strcmp(Prop,"u"))
        mat=&umat;
    else if (!strcmp(Prop,"visc"))
        mat=&viscmat;
    else
        printf("Parameter %s not found in LookupValue in R744.c\n",Prop);
    //At Low Temperature Index
    y1=(*mat)[iTlow][iPlow];
    y2=(*mat)[iTlow][iPhigh];
    y3=(*mat)[iTlow][iPhigh+1];
    P1=pvec[iPlow];
    P2=pvec[iPhigh];
    P3=pvec[iPhigh+1];
    a1=QuadInterpolate(P1,P2,P3,y1,y2,y3,p);
    //At High Temperature Index
    y1=(*mat)[iThigh][iPlow];
    y2=(*mat)[iThigh][iPhigh];
    y3=(*mat)[iThigh][iPhigh+1];
    a2=QuadInterpolate(P1,P2,P3,y1,y2,y3,p);
    //At High Temperature Index+1 (for QuadInterpolate() )
    y1=(*mat)[iThigh+1][iPlow];
    y2=(*mat)[iThigh+1][iPhigh];
    y3=(*mat)[iThigh+1][iPhigh+1];
    a3=QuadInterpolate(P1,P2,P3,y1,y2,y3,p);
    //At Final Interpolation
    T1=Tvec[iTlow];
    T2=Tvec[iThigh];
    T3=Tvec[iThigh+1];
    return QuadInterpolate(T1,T2,T3,a1,a2,a3,T);
}
static double powI(double x, int y)
{
    int i;
    double product=1.0;
    double x_in;
    int y_in;
    if (y==0)
    {
        return 1.0;
    }
    if (y<0)
    {
        x_in=1/x;
        y_in=-y;
    }
   else
   {
      x_in=x;
      y_in=y;
   }
    if (y_in==1)
    {
        return x_in;
    }
    product=x_in;
    for (i=1;i<y_in;i++)
    {
        product=product*x_in;
    }
    return product;
}
static double QuadInterpolate(double x0, double x1, double x2, double f0, double f1,
    double f2, double x)
{
    double L0, L1, L2;
    L0=((x-x1)*(x-x2))/((x0-x1)*(x0-x2));
    L1=((x-x0)*(x-x2))/((x1-x0)*(x1-x2));
    L2=((x-x0)*(x-x1))/((x2-x0)*(x2-x1));
```

```
    return L0*f0+L1*f1+L2*f2;
}
static int isNAN(double x)
{
   // recommendation from http://www.devx.com/tips/Tip/42853
   return x != x;
}
static int isINFINITY(double x)
{
   // recommendation from http://www.devx.com/tips/Tip/42853
   if ((x == x) && ((x - x) != 0.0))
      return 1;//return (x < 0.0 ? -1 : 1); // This will tell you whether positive or
           negative infinity
   else
      return 0;
}
```

R410A.h

```
#ifndef R410_H
#define R410_H
   double rho_R410A(double T, double p, int Types);
   double p_R410A(double T, double rho);
   double h_R410A(double T, double p_rho, int Types);
   double s_R410A(double T, double p_rho, int Types);
   double u_R410A(double T, double p_rho, int Types);
   double cp_R410A(double T, double p_rho, int Types);
   double cv_R410A(double T, double p_rho, int Types);
   double visc_R410A(double T, double p_rho, int Types);
   double k_R410A(double T, double p_rho, int Types);
   double w_R410A(double T, double p_rho, int Types);
   double MM_R410A(void);
   double Tcrit_R410A(void);
   double pcrit_R410A(void);
   double p_dp_R410A(double T);
   double p_bp_R410A(double T);
   double rhosatL_R410A(double T);
   double rhosatV_R410A(double T);
   // Derivatives
   double dhdrho_R410A(double T, double p_rho, int Types);
   double dhdT_R410A(double T, double p_rho, int Types);
   double dpdrho_R410A(double T, double p_rho, int Types);
   double dpdT_R410A(double T, double p_rho, int Types);
   int errCode_R410A(void);
#endif
```

R410A.c

```
/*
Properties for R410A.
by Ian Bell
Pseudo-pure fluid thermo props from
"Pseudo-pure fluid Equations of State for the Refrigerant Blends R410A, R404A, R507C
    and R407C"
by E.W. Lemmon, Int. J. Thermophys. v. 24, n4, 2003
In order to call the exposed functions, rho_, h_, s_, cp_,.....
there are three different ways the inputs can be passed, and this is expressed by the
     Types integer flag.
These macros are defined in the PropMacros.h header file:
1) First parameter temperature, second parameter pressure ex: h_R410A(260,354.7,1)
    =274
   In this case, the lookup tables are built if needed and then interpolated
2) First parameter temperature, second parameter density ex: h_R410A(260,13.03,2)=274
   Density and temp plugged directly into EOS
3) First parameter temperature, second parameter pressure ex: h_R410A(260,354.7,3)
    =274
   Density solved for, then plugged into EOS (can be quite slow)
*/
#if defined(_MSC_VER)
#define _CRTDBG_MAP_ALLOC
#define _CRT_SECURE_NO_WARNINGS
#include <stdlib.h>
#include <crtdbg.h>
```

```
#else
#include <stdlib.h>
#endif
#include "math.h"
#include "stdio.h"
#include <string.h>
#include "PropErrorCodes.h"
#include "PropMacros.h"
#include "R410A.h"
static int errCode;
static char errStr[ERRSTRLENGTH];
#define nP 200
#define nT 200
static double Tmin=150, Tmax=550, Pmin=70.03, Pmax=6000;
static double hmat[nT][nP];
static double rhomat[nT][nP];
static double cpmat[nT][nP];
static double smat[nT][nP];
static double cvmat[nT][nP];
static double umat[nT][nP];
static double viscmat[nT][nP];
static double kmat[nT][nP];
static double Tvec[nT];
static double pvec[nP];
static int TablesBuilt;
static const double c[]={
   2.8749,          //[0]
    2.0623,         //[1]
    5.9751,         //[2]
   1.5612           //[3]
};
static const double e[]={
    0.1,        //[0]
    697,        //[1]
    1723.0,         //[2]
    3875.0          //[3]
};
static const double a[]={
    36.8871,        //[0]
    7.15807,        //[1]
    -46.87575,      //[2]
    2.0623,         //[3]
    5.9751,         //[4]
    1.5612          //[5]
};
static const double b[]={
    0,              //[0]
    0,              //[1]
    -0.1,           //[2]
    2.02326,        //[3]
    5.00154,        //[4]
    11.2484         //[5]
};
static const double N[]={
    0.0,        //[0]
     0.987252,      //[1]
    -1.03017,       //[2]
     1.17666,       //[3]
    -0.138991,      //[4]
     0.00302373,    //[5]
    -2.53639,       //[6]
    -1.96680,       //[7]
    -0.830480,      //[8]
     0.172477,      //[9]
    -0.261116,      //[10]
    -0.0745473,         //[11]
     0.679757,      //[12]
    -0.652431,      //[13]
     0.0553849,         //[14]
    -0.0710970,         //[15]
    -0.000875332, //[16]
     0.0200760,         //[17]
    -0.0139761,         //[18]
    -0.0185110,         //[19]
     0.0171939,         //[20]
    -0.00482049         //[21]
};
static const double j[]={
   0.0,         //[0]
```

```
    0.44,           //[1]
    1.2,        //[2]
    2.97,           //[3]
    2.95,           //[4]
    0.2,        //[5]
    1.93,           //[6]
    1.78,           //[7]
    3.0,        //[8]
    0.2,        //[9]
    0.74,           //[10]
    3.0,        //[11]
    2.1,        //[12]
    4.3,        //[13]
    0.25,           //[14]
    7.0,        //[15]
    4.7,        //[16]
    13.0,           //[17]
    16.0,           //[18]
    25.0,           //[19]
    17.0,           //[20]
    7.4             //[21]
};
static const int i[]={
   0,           //[0]
    1,              //[1]
    1,              //[2]
    1,              //[3]
    2,              //[4]
    5,              //[5]
    1,              //[6]
    2,              //[7]
    3,              //[8]
    5,              //[9]
    5,              //[10]
    5,              //[11]
    1,              //[12]
    1,              //[13]
    4,              //[14]
    4,              //[15]
    9,              //[16]
    2,              //[17]
    2,              //[18]
    4,              //[19]
    5,              //[20]
    6           //[21]
};
static const int L[]={
    0,              //[0]
   0,           //[1]
    0,              //[2]
    0,              //[3]
    0,              //[4]
    0,              //[5]
    1,              //[6]
    1,              //[7]
    1,              //[8]
    1,              //[9]
    1,              //[10]
    1,              //[11]
    2,              //[12]
    2,              //[13]
    2,              //[14]
    2,              //[15]
    2,              //[16]
    3,              //[17]
    3,              //[18]
    3,              //[19]
    3,              //[20]
    3           //[21]
};
static const int g[]={
   0,           //[0]
   0,           //[1]
   0,           //[2]
   0,           //[3]
   0,           //[4]
   0,           //[5]
   1,           //[6]
   1,           //[7]
   1,           //[8]
```

```
    1,          //[9]
    1,          //[10]
    1,          //[11]
    1,          //[12]
    1,          //[13]
    1,          //[14]
    1,          //[15]
    1,          //[16]
    1,          //[17]
    1,          //[18]
    1,          //[19]
    1,          //[20]
    1,          //[21]
};
static const double Nbp[]={
    0.0,        //[0]
   -7.2818,     //[1]
     2.5093,        //[2]
     -3.2695,       //[3]
     -2.8022        //[4]
};
static const double tbp[]={
    0.0,        //[0]
     1.0,       //[1]
     1.8,       //[2]
     2.4,       //[3]
     4.9            //[4]
};
static const double Ndp[]={
     0.0,       //[0]
    -7.4411,    //[1]
     1.9883,        //[2]
     -2.4925,       //[3]
     -3.2633        //[4]
};
static const double tdp[]={
    0.0,        //[0]
     1.0,       //[1]
     1.6,       //[2]
     2.4,       //[3]
     5.0            //[4]
};
static const double R=0.114547443; //8.314472/72.5854;
static const double M=72.5824; //[g/mol]
static const double Tm=344.494; //[K]
static const double pm=4901.2; //[MPa--> kPa]
static const double pc=4810; //[MPa--> kPa] From (Calm 2007 HPAC Engineering)
static const double rhom=459.0300696; //6.324*M; //[mol/dm^3--> kg/m^3]

// Local function prototypes
static double Pressure_Trho(double T, double rho);
static double IntEnergy_Trho(double T, double rho);
static double Enthalpy_Trho(double T, double rho);
static double Entropy_Trho(double T, double rho);
static double SpecHeatV_Trho(double T, double rho);
static double SpecHeatP_Trho(double T, double rho);
static double Viscosity_Trho(double T, double rho);
static double Conductivity_Trho(double T, double rho);

static double get_Delta(double T, double p);
static double LookupValue(char *Prop,double T, double p);
static double powInt(double x, int y);
static double QuadInterp(double x0, double x1, double x2, double f0, double f1,
    double f2, double x);
static double a0(double tau, double delta);
static double da0_dtau(double tau, double delta);
static double d2a0_dtau2(double tau, double delta);
static double da0_ddelta(double tau, double delta);
static double dhdT(double tau, double delta);
static double dhdrho(double tau, double delta);
static double dpdT(double tau, double delta);
static double dpdrho(double tau, double delta);
static double ar(double tau, double delta);
static double dar_dtau(double tau,double delta);
static double d2ar_dtau2(double tau, double delta);
static double dar_ddelta(double tau,double delta);
static double d2ar_ddelta2(double tau,double delta);
```

```
static double d2ar_ddelta_dtau(double tau,double delta);
static void WriteLookup(void)
{
   int i,j;
   FILE *fp_h,*fp_s,*fp_rho,*fp_u,*fp_cp,*fp_cv,*fp_visc;
   fp_h=fopen("h.csv","w");
   fp_s=fopen("s.csv","w");
   fp_u=fopen("u.csv","w");
   fp_cp=fopen("cp.csv","w");
   fp_cv=fopen("cv.csv","w");
   fp_rho=fopen("rho.csv","w");
   fp_visc=fopen("visc.csv","w");
   // Write the pressure header row
   for (j=0;j<nP;j++)
   {
      fprintf(fp_h,",%0.12f",pvec[j]);
      fprintf(fp_s,",%0.12f",pvec[j]);
      fprintf(fp_rho,",%0.12f",pvec[j]);
      fprintf(fp_u,",%0.12f",pvec[j]);
      fprintf(fp_cp,",%0.12f",pvec[j]);
      fprintf(fp_cv,",%0.12f",pvec[j]);
      fprintf(fp_visc,",%0.12f",pvec[j]);
   }
   fprintf(fp_h,"\n");
   fprintf(fp_s,"\n");
   fprintf(fp_rho,"\n");
   fprintf(fp_u,"\n");
   fprintf(fp_cp,"\n");
   fprintf(fp_cv,"\n");
   fprintf(fp_visc,"\n");
   for (i=1;i<nT;i++)
   {
      fprintf(fp_h,"%0.12f",Tvec[i]);
      fprintf(fp_s,"%0.12f",Tvec[i]);
      fprintf(fp_rho,"%0.12f",Tvec[i]);
      fprintf(fp_u,"%0.12f",Tvec[i]);
      fprintf(fp_cp,"%0.12f",Tvec[i]);
      fprintf(fp_cv,"%0.12f",Tvec[i]);
      fprintf(fp_visc,"%0.12f",Tvec[i]);
      for (j=0;j<nP;j++)
      {
         fprintf(fp_h,",%0.12f",hmat[i][j]);
         fprintf(fp_s,",%0.12f",smat[i][j]);
         fprintf(fp_rho,",%0.12f",rhomat[i][j]);
         fprintf(fp_u,",%0.12f",umat[i][j]);
         fprintf(fp_cp,",%0.12f",cpmat[i][j]);
         fprintf(fp_cv,",%0.12f",cvmat[i][j]);
         fprintf(fp_visc,",%0.12f",viscmat[i][j]);
      }
      fprintf(fp_h,"\n");
      fprintf(fp_s,"\n");
      fprintf(fp_rho,"\n");
      fprintf(fp_u,"\n");
      fprintf(fp_cp,"\n");
      fprintf(fp_cv,"\n");
      fprintf(fp_visc,"\n");
   }
}
static void BuildLookup(void)
{
   int i,j;
   // Properties evaluated at all points with X in the
   // following p-h plot:
   /*       Supercritical
   ||X   X X X X  X X X X X X X X X X X X X X X X X
   ||X   X X X X  X X X X X X X X X X X X X X X X X
   ||       --------  X X X X X X X X X X X
   ||      /          \  X X X X X X X X X X X
p  ||     /         | X X X X Superheated Gas
   ||    /  Two     / X X X X X X X X X X X
   ||    /  Phase  /X X X X X X X X X X X X
   ||    /           / X X X X X X X X X X X
   ||=   = = = = = = = = = = = = = = = = = = = =
                 Enthalpy
   */
   if (!TablesBuilt)
```

```
    {
        printf("Building Lookup Tables... Please wait...\n");
        for (i=0;i<nT;i++)
        {
            Tvec[i]=Tmin+i*(Tmax-Tmin)/(nT-1);
        }
        for (j=0;j<nP;j++)
        {
            pvec[j]=Pmin+j*(Pmax-Pmin)/(nP-1);
        }
        for (i=0;i<nT;i++)
        {
            for (j=0;j<nP;j++)
            {
                if (Tvec[i]>Tm || pvec[j]>p_dp_R410A(Tvec[i]) || pvec[j]<p_bp_R410A(Tvec[
                    i]))
                {
                    rhomat[i][j]=get_Delta(Tvec[i],pvec[j])*rhom;
                    hmat[i][j]=h_R410A(Tvec[i],rhomat[i][j],TYPE_Trho);
                    smat[i][j]=s_R410A(Tvec[i],rhomat[i][j],TYPE_Trho);
                    umat[i][j]=u_R410A(Tvec[i],rhomat[i][j],TYPE_Trho);
                    cpmat[i][j]=cp_R410A(Tvec[i],rhomat[i][j],TYPE_Trho);
                    cvmat[i][j]=cv_R410A(Tvec[i],rhomat[i][j],TYPE_Trho);
                    viscmat[i][j]=visc_R410A(Tvec[i],rhomat[i][j],TYPE_Trho);
                    kmat[i][j]=k_R410A(Tvec[i],rhomat[i][j],TYPE_Trho);
                }
                else
                {
                    hmat[i][j]=_HUGE;
                    smat[i][j]=_HUGE;
                    umat[i][j]=_HUGE;
                    rhomat[i][j]=_HUGE;
                    umat[i][j]=_HUGE;
                    cpmat[i][j]=_HUGE;
                    cvmat[i][j]=_HUGE;
                    viscmat[i][j]=_HUGE;
                    kmat[i][j]=_HUGE;
                }
            }
        }
        TablesBuilt=1;
        //WriteLookup();
    }
}
double rho_R410A(double T, double p, int Types)
{
    errCode=0; // Reset error code
    switch(Types)
    {
        case TYPE_TPNoLookup:
            return get_Delta(T,p)*rhom;
        case TYPE_TP:
            BuildLookup();
            return LookupValue("rho",T,p);
        default:
            errCode=BAD_PROPCODE;
            return _HUGE;
    }
}
double p_R410A(double T, double rho)
{
    return Pressure_Trho(T,rho);
}
double h_R410A(double T, double p_rho, int Types)
{
    double rho;
    errCode=0; // Reset error code
    switch(Types)
    {
        case TYPE_Trho:
            return Enthalpy_Trho(T,p_rho);
        case TYPE_TPNoLookup:
            rho=get_Delta(T,p_rho)*rhom;
            return Enthalpy_Trho(T,rho);
        case TYPE_TP:
            BuildLookup();
            return LookupValue("h",T,p_rho);
        default:
            errCode=BAD_PROPCODE;
            return _HUGE;
    }
```

```
}
double s_R410A(double T, double p_rho, int Types)
{
   double rho;
   errCode=0; // Reset error code
   switch(Types)
   {
      case TYPE_Trho:
         return Entropy_Trho(T,p_rho);
      case TYPE_TPNoLookup:
         rho=get_Delta(T,p_rho)*rhom;
         return Entropy_Trho(T,rho);
      case TYPE_TP:
         BuildLookup();
         return LookupValue("s",T,p_rho);
      default:
         errCode=BAD_PROPCODE;
         return _HUGE;
   }
}
double u_R410A(double T, double p_rho, int Types)
{
   double rho;
   errCode=0; // Reset error code
   switch(Types)
   {
      case TYPE_Trho:
         return IntEnergy_Trho(T,p_rho);
      case TYPE_TPNoLookup:
         rho=get_Delta(T,p_rho)*rhom;
         return IntEnergy_Trho(T,rho);
      case TYPE_TP:
         BuildLookup();
         return LookupValue("u",T,p_rho);
      default:
         errCode=BAD_PROPCODE;
         return _HUGE;
   }
}
double cp_R410A(double T, double p_rho, int Types)
{
   double rho;
   errCode=0; // Reset error code
   switch(Types)
   {
      case TYPE_Trho:
         return SpecHeatP_Trho(T,p_rho);
      case TYPE_TPNoLookup:
         rho=get_Delta(T,p_rho)*rhom;
         return SpecHeatP_Trho(T,rho);
      case TYPE_TP:
         BuildLookup();
         return LookupValue("cp",T,p_rho);
      default:
         errCode=BAD_PROPCODE;
         return _HUGE;
   }
}
double cv_R410A(double T, double p_rho, int Types)
{
   double rho;
   errCode=0; // Reset error code
   switch(Types)
   {
      case TYPE_Trho:
         return SpecHeatV_Trho(T,p_rho);
      case TYPE_TPNoLookup:
         rho=get_Delta(T,p_rho)*rhom;
         return SpecHeatV_Trho(T,rho);
      case TYPE_TP:
         BuildLookup();
         return LookupValue("cv",T,p_rho);
      default:
         errCode=BAD_PROPCODE;
         return _HUGE;
   }
}
double visc_R410A(double T, double p_rho, int Types)
{
   double rho;
   errCode=0; // Reset error code
   switch(Types)
   {
```

```
        case TYPE_Trho:
            return Viscosity_Trho(T,p_rho);
        case TYPE_TPNoLookup:
            rho=get_Delta(T,p_rho)*rhom;
            return Viscosity_Trho(T,rho);
        case TYPE_TP:
            BuildLookup();
            return LookupValue("visc",T,p_rho);
        default:
            errCode=BAD_PROPCODE;
            return _HUGE;
    }
}
double k_R410A(double T, double p_rho, int Types)
{
    double rho;
    errCode=0; // Reset error code
    switch(Types)
    {
        case TYPE_Trho:
            return Conductivity_Trho(T,p_rho);
        case TYPE_TPNoLookup:
            rho=get_Delta(T,p_rho)*rhom;
            return Conductivity_Trho(T,rho);
        case TYPE_TP:
            BuildLookup();
            return LookupValue("k",T,p_rho);
        default:
            errCode=BAD_PROPCODE;
            return _HUGE;
    }
}
double w_R410A(double T, double p_rho, int Types)
{
    double rho;
    double delta,tau,c1,c2;
    errCode=0; // Reset error code
    switch(Types)
    {
        case TYPE_Trho:
            rho=p_rho; break;
        case TYPE_TPNoLookup:
            rho=get_Delta(T,p_rho)*rhom; break;
        default:
            errCode=BAD_PROPCODE;
            return _HUGE;
    }
    delta=rho/rhom;
    tau=Tm/T;
    c1=-SpecHeatV_Trho(T,rho)/R;
    c2=(1.0+2.0*delta*dar_ddelta(tau,delta)+delta*delta*d2ar_ddelta2(tau,delta));
     return sqrt(-c2*T*SpecHeatP_Trho(T,rho)*1000/c1);
}
double pcrit_R410A(void)
{
    return pm;
}
double Tcrit_R410A(void)
{
    return Tm;
}
double MM_R410A(void)
{
    return M;
}
int errCode_R410A(void)
{
    return errCode;
}
double p_bp_R410A(double T)
{
    //Bubble point of R410A
    double sum=0,theta;
    int k;
     theta=1-T/Tm;
     for (k=1;k<=4;k++)
     {
         sum+=Nbp[k]*pow(theta,tbp[k]);
     }
     return pm*exp(Tm/T*sum);
```

```
}
double p_dp_R410A(double T)
{
   //Dew point of R410A
   double sum=0,theta;
   int k;
    theta=1-T/Tm;
    for (k=1;k<=4;k++)
    {
        sum+=Ndp[k]*pow(theta,tdp[k]);
    }
    return pm*exp(Tm/T*sum);
}
double rhosatV_R410A(double T)
{
   double THETA,a1,a2,a3,a4,a5,a6,a7,a8,b1,b2,b3,b4,b5,b6,b7,b8;
   THETA=1-T/344.5;
   a1 =        -4.02;
   a2 =        -18.8;
   a3 =       -18.16;
   a4 =       -17.37;
   a5 =       -16.15;
   a6 =        -7.87;
   a7 =       -20.46;
   a8 =       -18.59;
   b1 =       0.4803;
   b2 =        9.217;
   b3 =         7.52;
   b4 =        3.471;
   b5 =        5.124;
   b6 =        1.498;
   b7 =        9.831;
   b8 =         8.99;
   return exp(a1*pow(THETA,b1)+a2*pow(THETA,b2)+a3*pow(THETA,b3)+a4*pow(THETA,b4)+a5*
       pow(THETA,b5)+a6*pow(THETA,b6)+a7*pow(THETA,b7)+a8*pow(THETA,b8))*459.53;
}
double rhosatL_R410A(double T)
{
   double THETA,a1,a2,a3,a4,a5,a6,b1,b2,b3,b4,b5,b6;
   THETA=1-T/344.5;
   a1 =        2.876;
   a2 =       -2.465;
   a3 =      -0.9235;
   a4 =      -0.4798;
   a5 =       0.7394;
   a6 =        1.762;
   b1 =        0.365;
   b2 =       0.6299;
   b3 =        1.864;
   b4 =        1.932;
   b5 =        2.559;
   b6 =        1.149;
   return exp(a1*pow(THETA,b1)+a2*pow(THETA,b2)+a3*pow(THETA,b3)+a4*pow(THETA,b4)+a5*
       pow(THETA,b5)+a6*pow(THETA,b6))*459.53;
}
double Viscosity_Trho(double T, double rho)
{
    // Properties taken from "Viscosity of Mixed
   // Refrigerants R404A,R407C,R410A, and R507A"
   // by Vladimir Geller,
   // 2000 Purdue Refrigeration conferences
    // inputs in T [K], and p [kPa]
    // output in Pa-s
   double eta_microPa_s;
   //Set constants required
   double a_0=-2.695e0,a_1=5.850e-2,a_2=-2.129e-5,b_1=9.047e-3,b_2=5.784e-5,
      b_3=1.309e-7,b_4=-2.422e-10,b_5=9.424e-14,b_6=3.933e-17;
   eta_microPa_s=a_0+a_1*T+a_2*T*T+b_1*rho+b_2*rho*rho+b_3*rho*rho*rho+b_4*rho*rho*
       rho*rho+b_5*rho*rho*rho*rho*rho+b_6*rho*rho*rho*rho*rho*rho;
   return eta_microPa_s/1e6;
}
double Conductivity_Trho(double T, double rho)
{
   // Properties taken from "Thermal Conductivity
   // of the Refrigerant mixtures R404A,R407C,R410A, and R507A"
   // by V.Z. Geller, B.Z. Nemzer, and U.V. Cheremnykh
   // Int. J. Thermophysics, v. 22, n 4 2001
   // inputs in T [K], and p [kPa] or rho [kg/m^3]
```

```
    // output in W/m-K
    //Set constants required
    double a_0=-8.872e0,a_1=7.410e-2,b_1=3.576e-2,b_2=-9.045e-6,b_3=4.343e-8,b_4
        =-3.705e-12;
    return (a_0+a_1*T+b_1*rho+b_2*rho*rho+b_3*rho*rho*rho+b_4*rho*rho*rho*rho)/1.e6;
        // from mW/m-K to kW/m-K
}
double dhdT_R410A(double T, double p_rho, int Types)
{
    double rho;
    errCode=0; // Reset error code
    switch(Types)
    {
        case TYPE_Trho:
            rho=p_rho;
            return dhdT(Tm/T,rho/rhom);
        case TYPE_TPNoLookup:
            rho=get_Delta(T,p_rho)*rhom;
            return dhdT(Tm/T,rho/rhom);
        case 99:
            rho=get_Delta(T,p_rho)*rhom;
            return (Enthalpy_Trho(T+0.001,rho)-Enthalpy_Trho(T,rho))/0.001;
        default:
            errCode=BAD_PROPCODE;
            return _HUGE;
    }
}
double dhdrho_R410A(double T, double p_rho, int Types)
{
    double rho;
    errCode=0; // Reset error code
    switch(Types)
    {
        case TYPE_Trho:
            rho=p_rho;
            return dhdrho(Tm/T,rho/rhom);
        case TYPE_TPNoLookup:
            rho=get_Delta(T,p_rho)*rhom;
            return dhdrho(Tm/T,rho/rhom);
        case 99:
            rho=get_Delta(T,p_rho)*rhom;
            return (Enthalpy_Trho(T,rho+0.001)-Enthalpy_Trho(T,rho))/0.001;
        default:
            errCode=BAD_PROPCODE;
            return _HUGE;
    }
}
double dpdT_R410A(double T, double p_rho, int Types)
{
    double rho;
    errCode=0; // Reset error code
    switch(Types)
    {
        case TYPE_Trho:
            rho=p_rho;
            return dpdT(Tm/T,rho/rhom);
        case TYPE_TPNoLookup:
            rho=get_Delta(T,p_rho)*rhom;
            return dpdT(Tm/T,rho/rhom);
        case 99:
            rho=get_Delta(T,p_rho)*rhom;
            return (Pressure_Trho(T+0.01,rho)-Pressure_Trho(T,rho))/0.01;
        default:
            errCode=BAD_PROPCODE;
            return _HUGE;
    }
}
double dpdrho_R410A(double T, double p_rho, int Types)
{
    double rho;
    errCode=0; // Reset error code
    switch(Types)
    {
        case TYPE_Trho:
            rho=p_rho;
            return dpdrho(Tm/T,rho/rhom);
        case TYPE_TPNoLookup:
            rho=get_Delta(T,p_rho)*rhom;
            return dpdrho(Tm/T,rho/rhom);
        case 99:
```

```
            rho=get_Delta(T,p_rho)*rhom;
            return (Pressure_Trho(T,rho+0.0001)-Pressure_Trho(T,rho))/0.0001;
        default:
            errCode=BAD_PROPCODE;
            return _HUGE;
    }
}
/***********************************************/
/*           Property Derivatives              */
/***********************************************/
// See Lemmon, 2000 for more information
static double dhdrho(double tau, double delta)
{
    double T;
    T=Tm/tau;
    //Note: dphi02_dDelta_dTau(tau,delta) is equal to zero
    return R*T/rhom*(tau*(d2ar_ddelta_dtau(tau,delta))+dar_ddelta(tau,delta)+delta*
        d2ar_ddelta2(tau,delta));
}
static double dhdT(double tau, double delta)
{
    double dhdT_rho,T,dhdtau;
    T=Tm/tau;
    dhdT_rho=R*tau*(da0_dtau(tau,delta)+dar_dtau(tau,delta))+R*delta*dar_ddelta(tau,
        delta)+R;
    dhdtau=R*T*(da0_dtau(tau,delta)+ dar_dtau(tau,delta))+R*T*tau*(d2a0_dtau2(tau,
        delta)+d2ar_dtau2(tau,delta))+R*T*delta*d2ar_ddelta_dtau(tau,delta);
    return dhdT_rho+dhdtau*(-Tm/T/T);
}
static double dpdT(double tau, double delta)
{
    double T,rho;
    T=Tm/tau;     rho=delta*rhom;
    return rho*R*(1+delta*dar_ddelta(tau,delta)-delta*tau*d2ar_ddelta_dtau(tau,delta))
        ;
}
static double dpdrho(double tau, double delta)
{
    double T,rho;
    T=Tm/tau;    rho=delta*rhom;
    return R*T*(1+2*delta*dar_ddelta(tau,delta)+delta*delta*d2ar_ddelta2(tau,delta));
}

static double get_Delta(double T, double p)
{
     double change,eps=1e-8, tau,delta_guess;
     int counter=1;
     double r1,r2,r3,delta1,delta2,delta3;
    if (T>Tm)
    {
        delta_guess=p/(8.314/M*T)/rhom/0.7; //0.7 for compressibility factor
    }
    else
    {
       if (p<=(0.5*p_dp_R410A(T)+0.5*p_bp_R410A(T)))
       {
          // Superheated vapor
          delta_guess=p/(8.314/M*T)/rhom;
       }
       else
       {
          // Subcooled liquid
          delta_guess=10;
       }
    }

    tau=Tm/T;
     delta1=delta_guess;
     delta2=delta_guess+.001;
     r1=p/(delta1*rhom*R*T)-1.0-delta1*dar_ddelta(tau,delta1);
     r2=p/(delta2*rhom*R*T)-1.0-delta2*dar_ddelta(tau,delta2);
     while(counter==1 || fabs(change)>eps)
     {
         delta3=delta2-r2/(r2-r1)*(delta2-delta1);
         r3=p/(delta3*rhom*R*T)-1.0-delta3*dar_ddelta(tau,delta3);
         change=r2/(r2-r1)*(delta2-delta1);
         delta1=delta2;
         delta2=delta3;
         r1=r2;
         r2=r3;
         counter=counter+1;
         /*mexPrintf("%g \t %g \t %g \t %g \t %g \n",delta1,r1,delta2,r2,change);*/
```

```
    }
    return delta3;
}
// ****************************************
//          THERMODYNAMIC FUNCTIONS
// ****************************************
static double Pressure_Trho(double T, double rho)
{
   double delta,tau;
   delta=rho/rhom;
   tau=Tm/T;
   return R*T*rho*(1.0+delta*dar_ddelta(tau,delta));
}
static double IntEnergy_Trho(double T, double rho)
{
   double delta,tau;
   delta=rho/rhom;
   tau=Tm/T;
   return R*T*tau*(da0_dtau(tau,delta)+dar_dtau(tau,delta));
}
static double Enthalpy_Trho(double T, double rho)
{
   double delta,tau;
   delta=rho/rhom;
   tau=Tm/T;
   return R*T*(1.0+tau*(da0_dtau(tau,delta)+dar_dtau(tau,delta))+delta*dar_ddelta(tau
       ,delta));
}
static double Entropy_Trho(double T, double rho)
{
   double delta,tau;
   delta=rho/rhom;
   tau=Tm/T;
   return R*(tau*(da0_dtau(tau,delta)+dar_dtau(tau,delta))-a0(tau,delta)-ar(tau,delta
       ));
}
static double SpecHeatV_Trho(double T, double rho)
{
   double delta,tau;
   delta=rho/rhom;
   tau=Tm/T;
   return -R*tau*tau*(d2a0_dtau2(tau,delta)+d2ar_dtau2(tau,delta));
}
static double SpecHeatP_Trho(double T, double rho)
{
   double delta,tau,c1,c2;
   delta=rho/rhom;
   tau=Tm/T;
   c1=1.0+delta*dar_ddelta(tau,delta)-delta*tau*d2ar_ddelta_dtau(tau,delta);
   c2=1.0+2.0*delta*dar_ddelta(tau,delta)+delta*delta*d2ar_ddelta2(tau,delta);
   return R*(-tau*tau*(d2a0_dtau2(tau,delta)+d2ar_dtau2(tau,delta))+c1*c1/c2);
}
// ************************************************
//              HELMHOLTZ DERIVATIVES
// ************************************************
static double a0(double tau, double delta)
{
   double sum;
   int k;
   sum=log(delta)-log(tau)+a[0]+a[1]*tau+a[2]*pow(tau,b[2]);
   for(k=3;k<=5;k++)
   {
      sum+=a[k]*log(1.0-exp(-b[k]*tau));
   }
   return sum;
}
static double da0_dtau(double tau, double delta)
{
   double sum;
   int k;
   sum=-1/tau+a[1]+a[2]*b[2]*pow(tau,b[2]-1);
   for(k=3;k<=5;k++)
   {
      sum+=a[k]*b[k]*exp(-b[k]*tau)/(1-exp(-b[k]*tau));
   }
   return sum;
}
static double d2a0_dtau2(double tau, double delta)
{
```

```
    double sum;
    int k;
    sum=1.0/(tau*tau)+a[2]*b[2]*(b[2]-1)*pow(tau,b[2]-2);
    for(k=3;k<=5;k++)
    {
       sum+=-a[k]*b[k]*b[k]/(4.0*powInt(sinh(b[k]*tau/2.0),2));
    }
    return sum;
}
static double da0_ddelta(double tau, double delta)
{
    return 1/delta;
}
static double ar(double tau, double delta)
{
    double sum=0;
    int k;
    for  (k=1;k<=21;k++)
     {
         sum+=N[k]*pow(delta,i[k])*pow(tau,j[k])*exp(-g[k]*pow(delta,L[k]));
     }
    return sum;
}
static double dar_dtau(double tau,double delta)
{
    double sum=0;
    int k;
    for  (k=1;k<=21;k++)
     {
         sum+=j[k]*N[k]*pow(delta,i[k])*pow(tau,j[k]-1)*exp(-g[k]*pow(delta,L[k]));
     }
    return sum;
}
static double d2ar_dtau2(double tau, double delta)
{
    double sum=0;
    int k;
    for  (k=1;k<=21;k++)
     {
         sum+=N[k]*pow(delta,i[k])*j[k]*(j[k]-1)*pow(tau,j[k]-2)*exp(-g[k]*pow(delta,L
             [k]));
     }
    return sum;
}
static double d2ar_ddelta2(double tau,double delta)
{
     double dar2_dDelta2=0;
     int k;
     for  (k=1;k<=21;k++)
     {
         dar2_dDelta2=dar2_dDelta2+N[k]*pow(tau,j[k])*exp(-g[k]*pow(delta,L[k]))*(pow(
             delta,i[k]-2)*pow(i[k],2)-pow(delta,i[k]-2)*i[k]-2*pow(delta,i[k]-2+L[k])
             *i[k]*g[k]*L[k]-pow(delta,i[k]-2+L[k])*g[k]*pow(L[k],2)+pow(delta,i[k]-2+
             L[k])*g[k]*L[k]+pow(delta,i[k]+2*L[k]-2)*pow(g[k],2)*pow(L[k],2));
     }
     return dar2_dDelta2;
}
static double dar_ddelta(double tau,double delta)
{
     double sum=0,gk,ik,Lk;
     int k;
     for  (k=1;k<=21;k++)
     {
       gk=(double)g[k];
       ik=(double)i[k];
       Lk=(double)L[k];
       sum+=N[k]*powInt(delta,i[k]-1)*pow(tau,j[k])*exp(-gk*powInt(delta,L[k]))*(ik-
           powInt(delta,L[k])*gk*Lk);
     }
     return sum;
}
static double d2ar_ddelta_dtau(double tau,double delta)
{
     double dar_dDelta_dTau=0;
     int k;
     for  (k=1;k<=21;k++)
     {
         dar_dDelta_dTau=dar_dDelta_dTau+N[k]*j[k]*pow(tau,j[k]-1)*(i[k]*pow(delta,i[k
             ]-1)*exp(-g[k]*pow(delta,L[k]))+pow(delta,i[k])*exp(-g[k]*pow(delta,L[k])
             )*(-g[k]*L[k]*pow(delta,L[k]-1)));
     }
```

```
    return dar_dDelta_dTau;
}
// ********************************************************
//                    MAINTENANCE FUNCTIONS
// ********************************************************
static double LookupValue(char *Prop, double T, double p)
{
   int iPlow, iPhigh, iTlow, iThigh,L,R,M;
   double T1, T2, T3, P1, P2, P3, y1, y2, y3, a1, a2, a3;
   double (*mat)[nT][nP];
   if (T>Tmax || T<Tmin)
   {
      errCode=OUT_RANGE_T;
   }
   if (p>Pmax || p<Pmin)
   {
      errCode=OUT_RANGE_P;
   }
   if (T>Tmax || T<Tmin || p>Pmax ||p<Pmin)
   {
      printf("Input to LookupValue() for %s is out of bounds [T:%g p:%g]\n",Prop,T,p)
          ;
      return -1e6;
   }
   L=0;
   R=nP-1;
   M=(L+R)/2;
   // Use interval halving to find the indices which bracket the temperature of
       interest
   while (R-L>1)
   {
      if (T>=Tvec[M])
      { L=M; M=(L+R)/2; }
      if (T<Tvec[M])
      { R=M; M=(L+R)/2; }
   }
   iTlow=L; iThigh=R;
   L=0;
   R=nP-1;
   M=(L+R)/2;
   // Use interval halving to find the indices which bracket the pressure of interest
   while (R-L>1)
   {
      if (p>=pvec[M])
      { L=M; M=(L+R)/2; }
      if (p<pvec[M])
      { R=M; M=(L+R)/2; }
   }
   iPlow=L; iPhigh=R;
   /* Depending on which property is desired,
   make the matrix "mat" a pointer to the
   desired property matrix */
   if (!strcmp(Prop,"rho"))
      mat=&rhomat;
   if (!strcmp(Prop,"cp"))
      mat=&cpmat;
   if (!strcmp(Prop,"cv"))
      mat=&cvmat;
   if (!strcmp(Prop,"h"))
      mat=&hmat;
   if (!strcmp(Prop,"s"))
      mat=&smat;
   if (!strcmp(Prop,"u"))
      mat=&umat;
   if (!strcmp(Prop,"visc"))
      mat=&viscmat;
   if (!strcmp(Prop,"k"))
      mat=&kmat;
   //At Low Temperature Index
   y1=(*mat)[iTlow][iPlow];
   y2=(*mat)[iTlow][iPhigh];
   y3=(*mat)[iTlow][iPhigh+1];
   P1=pvec[iPlow];
   P2=pvec[iPhigh];
   P3=pvec[iPhigh+1];
   a1=QuadInterp(P1,P2,P3,y1,y2,y3,p);
   //At High Temperature Index
   y1=(*mat)[iThigh][iPlow];
   y2=(*mat)[iThigh][iPhigh];
```

```
    y3=(*mat)[iThigh][iPhigh+1];
    a2=QuadInterp(P1,P2,P3,y1,y2,y3,p);
    //At High Temperature Index+1 (for QuadInterp() )
    y1=(*mat)[iThigh+1][iPlow];
    y2=(*mat)[iThigh+1][iPhigh];
    y3=(*mat)[iThigh+1][iPhigh+1];
    a3=QuadInterp(P1,P2,P3,y1,y2,y3,p);
    //At Final Interpolation
    T1=Tvec[iTlow];
    T2=Tvec[iThigh];
    T3=Tvec[iThigh+1];
    return QuadInterp(T1,T2,T3,a1,a2,a3,T);
}
static double powInt(double x, int y)
{
    // Raise a double to an integer power
    // Overload not provided in math.h
     int i;
     double product=1.0;
     double x_in;
     int y_in;
     if (y==0)
     {
         return 1.0;
     }
     if (y<0)
     {
         x_in=1/x;
         y_in=-y;
     }
    else
    {
       x_in=x;
       y_in=y;
    }
     if (y_in==1)
     {
         return x_in;
     }
     product=x_in;
     for (i=1;i<y_in;i++)
     {
         product=product*x_in;
     }
     return product;
}
static double QuadInterp(double x0, double x1, double x2, double f0, double f1,
    double f2, double x)
{
    /* Quadratic interpolation.
    Based on method from Kreyszig,
    Advanced Engineering Mathematics, 9th Edition
    */
     double L0, L1, L2;
     L0=((x-x1)*(x-x2))/((x0-x1)*(x0-x2));
     L1=((x-x0)*(x-x2))/((x1-x0)*(x1-x2));
     L2=((x-x0)*(x-x1))/((x2-x0)*(x2-x1));
     return L0*f0+L1*f1+L2*f2;
}
```

MyFuncs.h

```
#include "StructsMacros.h"
#ifndef MY_FUNCS
#define MY_FUNCS
    char DisplayMode[100];
    // ****************************************************************
    // ****************** Function Prototypes *************************
    // ****************************************************************
    void swap(double *x, double *y);
    double powInt(double x, int y);
    double QuadInterp(double x0, double x1, double x2, double f0, double f1, double f2
        , double x);
    double * cross(double * a, double * b);
    double norm(double * x,int n);
```

```
    void bubbleSort(double *a,int MAX);
    double *linspace(double min, double max, int N);
    void append(double *vec1,int N1,double *vec2, int N2,double *vec3, int *N3);
    int Matrix2csv(char *fileName, double *Mat,int nR, int nC);
    int flowVec2csv(char *fileName,struct scrollVals *scroll,int nTheta);
    double trapz(double *x, double *y, int N);
    double acosh(double x);
    int makeDir(char * fName);
    int printf_plus(const char * fmt,...);
    int isNAN(double x);
    int isINFINITY(double x);
    double *colSlice(double *mat,int nR, int nC, int iC);
    double *rowSlice(double *mat,int nR, int nC, int iR);
    double sumVector(double *vec,int N);
#endif
```

MyFuncs.c

```
#ifndef __GNUC__
#define _CRT_SECURE_NO_WARNINGS
#define _CRTDBG_MAP_ALLOC
#include <stdlib.h>
#include <crtdbg.h>
#else
#include <stdlib.h>
#endif
#include <stdio.h>
#include <stdarg.h>
#include <string.h>

#include "math.h"
#include "StructsMacros.h"
#include "Geo/geoFuncs.h"
#include "ScrollsModel.h"
#include "MyFuncs.h"
#include "SaveOutputs.h"

static void outputCV(FILE *fp,int CV,struct geoVals *geo, double theta)
{
    //Output the name of control volume to file given by file pointer
    int i;
    if (CV==Isa)
        fprintf(fp,"Isa");
    if (CV==Is1)
        fprintf(fp,"Is1");
    if (CV==Is2)
        fprintf(fp,"Is2");
    if (CV==Id1)
        fprintf(fp,"Id1");
    if (CV==Id2)
        fprintf(fp,"Id2");
    if (CV==Idd)
        fprintf(fp,"Idd");
    if (CV==Iddd)
        fprintf(fp,"Iddd");
    if (CV==Idischarge)
        fprintf(fp,"Idischarge");
    if (CV==Isuction)
        fprintf(fp,"Isuction");

    for (i=0;i<nC(geo,theta);i++)
    {
        if (CV==Ic1[i])
            fprintf(fp,"Ic1[%d]",i+1);
        if (CV==Ic2[i])
            fprintf(fp,"Ic2[%d]",i+1);
    }
}
double *colSlice(double *mat,int nR, int nC, int iC)
{
    // Take a column slice of a 2D matrix expressed with a single index
    int i;
    double *out;
    out=(double *)calloc(nR,sizeof(double));
    for (i=0;i<nR;i++)
    {
        out[i]=mat[i+nR*iC];
    }
    return out;
}
double *rowSlice(double *mat,int nR, int nC, int iR)
```

```
{
   // Take a row slice of a 2D matrix expressed with a single index
   int j;
   double *out;
   out=(double *)calloc(nC,sizeof(double));
   for (j=0;j<nC;j++)
   {
      out[j]=mat[iR+nR*j];
   }
   return out;
}
double sumVector(double *vec,int N)
{
   // Sum up all the elements of a vector
   int i;
   double s=0.0;
   for (i=0;i<N;i++)
   {
      s=s+vec[i];
   }
   return s;
}
static void outputFlowModel(FILE *fp,int flowModel)
{
   // Print out flowModel to file given by fp
   if (flowModel==DRY_GAS_RADIAL_FRICTIONAL_MODEL)
      fprintf(fp,"DRY_GAS_RADIAL_FRICTIONAL_MODEL");
   if (flowModel==DRY_GAS_FLANK_FRICTIONAL_MODEL)
      fprintf(fp,"DRY_GAS_FLANK_FRICTIONAL_MODEL");
   if (flowModel==TEE_FLOW_MODEL)
      fprintf(fp,"TEE_FLOW_MODEL");
   if (flowModel==BENDS_MODEL)
      fprintf(fp,"BENDS_MODEL");
   if (flowModel==TWO_PHASE_NOZZLE)
      fprintf(fp,"TWO_PHASE_NOZZLE");
   if (flowModel==DRY_GAS_FLANK_FLANK_MODEL)
      fprintf(fp,"DRY_GAS_FLANK_FLANK_MODEL");
   if (flowModel==CORRECTED_RADIAL_NOZZLE)
      fprintf(fp,"CORRECTED_RADIAL_NOZZLE");
   if (flowModel==CORRECTED_FLANK_NOZZLE)
      fprintf(fp,"CORRECTED_FLANK_NOZZLE");
   if (flowModel==LIQUID_RADIAL_FRICTIONAL_MODEL)
      fprintf(fp,"LIQUID_RADIAL_FRICTIONAL_MODEL");
   if (flowModel==LIQUID_FLANK_FRICTIONAL_MODEL)
      fprintf(fp,"LIQUID_FLANK_FRICTIONAL_MODEL");
}
double * cross(double * a, double * b)
{
   // cross product of two vectors
   // a and b MUST be 3 element arrays
   //
   //          | i  j  k  |
   //  axb   = | ax ay az |
   //          | bx by bz |
   //
   // Vector Values
   //
   // ax=a[0],  bx=b[0]
   // ay=a[1],  by=b[1]
   // az=a[2],  bz=b[2]
   double *out;
   out=(double *)calloc(3,sizeof(double));
   out[0]=a[1]*b[2]-b[1]*a[2];
   out[1]=-(a[0]*b[2]-b[0]*a[2]);
   out[2]=a[0]*b[1]-b[0]*a[1];
   return out;
}
double norm(double * x,int n)
{
   int i;
   double sum=0;
   for (i=0;i<n;i++)
   {
      sum+=x[i]*x[i];
   }
   return sqrt(sum);
}
double powInt(double x, int y)
```

```
{
   // Raise a double value to an integer power
    int i;
    double product=1.0;
    double x_in;
    int y_in;
    if (y==0)
        return 1.0;
    else if (y<0)
   {
        x_in=1/x;
        y_in=-y;
    }
   else
   {
      x_in=x;
      y_in=y;
   }
    if (y_in==1)
    {
        return x_in;
    }
    product=x_in;
    for (i=1;i<y_in;i++)
    {
        product=product*x_in;
    }
    return product;
}
void append(double *vec1,int N1,double *vec2, int N2,double *vec3, int *N3)
{
   int i;
   *N3=N1+N2;
   for (i=0;i<N1;i++)
   {
      vec3[i]=vec1[i];
   }
   for (i=0;i<N2;i++)
   {
      vec3[i+N1]=vec2[i];
   }
}
double *linspace(double min, double max, int N)
{
   //C equivalent of MATLAB function linspace
   double *A; //Return pointer to matrix
   double dx;
   int i;
   if (N>1)
   {
      //"Vector" with a multiple elements
      dx=(max-min)/((double)(N-1));
   }
   else
   {
      //"Vector" with a single element
      dx=0;
   }
   A=(double*) calloc (N,sizeof(double));
   for (i=0;i<N;i++)
   {
      //Load up vector
      A[i]=min+dx*i;
   }
   return A;
}
void bubbleSort(double *a,int MAX)
{
   // Implementation of bubble sort algorithm
   // Currently not used
   int x,y;
   double t;
   for (x=0; x < MAX-1; x++)
   {
      for (y=0; y < MAX-x-1; y++)
      {
         if (a[y] > a[y+1])
         {
            t=a[y];
            a[y]=a[y+1];
```

```
            a[y+1]=t;
         }
      }
   }
}
double QuadInterp(double x0, double x1, double x2, double f0, double f1, double f2,
    double x)
{
   // Quadratic Lagrangian interpolation
    double L0, L1, L2;
    L0=((x-x1)*(x-x2))/((x0-x1)*(x0-x2));
    L1=((x-x0)*(x-x2))/((x1-x0)*(x1-x2));
    L2=((x-x0)*(x-x1))/((x2-x0)*(x2-x1));
    return L0*f0+L1*f1+L2*f2;
}
int Matrix2csv(char *fileName, double *Mat,int nR, int nC)
{
   // Output a matrix to csv file
   int i,j;
   FILE *fp;
   fp=fopen(fileName,"wb");
   if (fp==NULL)
   {
      fprintf(stderr,"Could not open file\n\n");
      exit(-1);
   }
   for (i=0;i<nR;i++)
   {
      for (j=0;j<nC;j++)
      {
         fprintf(fp,"%0.12e",Mat[i+nR*j]);
         if (j<nC-1)
            fprintf(fp,",");
      }
      if (i<nR-1)
      {
         fprintf(fp,"\n");
      }
   }
   fclose(fp);
   return 0;
}
int flowVec2csv(char *fileName,struct scrollVals *scroll,int nTheta)
{
   int i,j;
   FILE *fp;
   fp=fopen(fileName,"wb");
   if (fp==NULL)
   {
      fprintf(stderr,"Could not open file\n\n");
      exit(-1);
   }
   for (i=0;i < nTheta; i++)
   {
      fprintf(fp,"theta=, %g\n",scroll->theta[i]);
      fprintf(fp,"CV1,CV2,CVup,A,flowModel,h_up,T_up,p_up,p_down,xL,mdot,Re,Ma\n");
      for (j=0;j< scroll->flowVec[i].N;j++)
      {
         outputCV(fp,scroll->flowVec[i].CV1[j],&(scroll->geo),scroll->theta[i]);
         fprintf(fp,",");
         outputCV(fp,scroll->flowVec[i].CV2[j],&(scroll->geo),scroll->theta[i]);
         fprintf(fp,",");
         outputCV(fp,scroll->flowVec[i].CVup[j],&(scroll->geo),scroll->theta[i]);
         fprintf(fp,",");
         fprintf(fp,"%0.12e",scroll->flowVec[i].A[j]);
         fprintf(fp,",");
         outputFlowModel(fp,scroll->flowVec[i].flowModel[j]);
         fprintf(fp,",");
         fprintf(fp,"%0.12e",scroll->flowVec[i].h_up[j]);
         fprintf(fp,",");
         fprintf(fp,"%0.12e",scroll->flowVec[i].T_up[j]);
         fprintf(fp,",");
         fprintf(fp,"%0.12e",scroll->flowVec[i].p_up[j]);
         fprintf(fp,",");
         fprintf(fp,"%0.12e",scroll->flowVec[i].p_down[j]);
         fprintf(fp,",");
         fprintf(fp,"%0.12e",scroll->flowVec[i].xL[j]);
```

```
            fprintf(fp,",");
            fprintf(fp,"%0.12e",scroll->flowVec[i].mdot[j]);
            fprintf(fp,",");
            fprintf(fp,"%0.12e",scroll->flowVec[i].Re[j]);
            fprintf(fp,",");
            fprintf(fp,"%0.12e",scroll->flowVec[i].Ma[j]);
            fprintf(fp,"\n");
         }
         fprintf(fp,"\n\n");
      }
      fclose(fp);
      return 0;
}
double trapz(double *x, double *y, int N)
{
   // Numerical trapezoidal integration
   double sum=0;
   int i;
   for (i=0;i<N-1;i++)
   {
      sum+=(y[i]+y[i+1])/2.0*(x[i+1]-x[i]);
   }
   return sum;
}
double acosh(double x)
{
   return log(x + sqrt(x*x - 1.0) );
}
void swap(double *x, double *y)
{
    double temp;
    temp = *x;
    *x = *y;
    *y = temp;
}
int makeDir(char * fName)
{
#if defined(__WIN32__) || defined(__WIN64__) || defined(_MSC_VER)
   return mkdir(fName);
#else
   mkdir(fName,0777);
#endif
}
int printf_plus(const char * fmt,...)
{
   // from http://www.eumus.edu.uy/eme/c/c-notes_summit/intermediate/sx11.html
   va_list argp;
   FILE *fp;
   char str[1000];
   if (!strcmp(DisplayMode,"Terminal-File"))
   {
      BuildBatchPath();
      sprintf(str,"%s/log.txt",batchPath);
      fp=fopen(str,"a");
      va_start(argp, fmt); //open the variable list
      vfprintf(fp, fmt, argp);
      va_end(argp);
      fclose(fp);
      va_start(argp,fmt);
      vfprintf(stdout,fmt, argp);
      va_end(argp);
   }
   else
   {
      va_start(argp, fmt);
      vprintf(fmt, argp);
      va_end(argp);
   }
   return 1;
}
int isNAN(double x)
{
   // recommendation from http://www.devx.com/tips/Tip/42853
   return x != x;
}
int isINFINITY(double x)
```

```
{
   // recommendation from http://www.devx.com/tips/Tip/42853
   if ((x == x) && ((x - x) != 0.0))
      return 1;//return (x < 0.0 ? -1 : 1); // This will tell you whether positive or
           negative infinity
   else
      return 0;
}
```

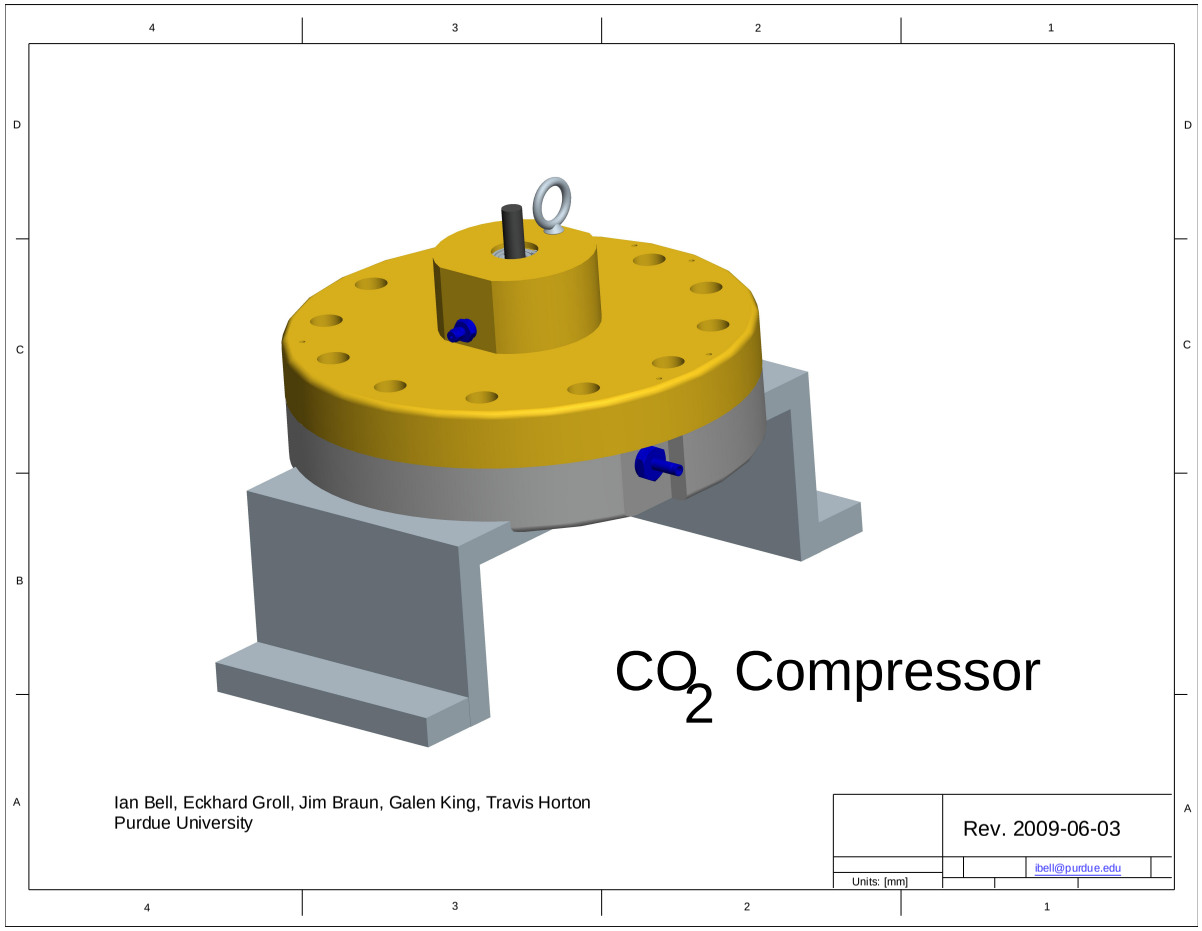


Figure G.1. Drawings of Prototype CO_2 Compressor.

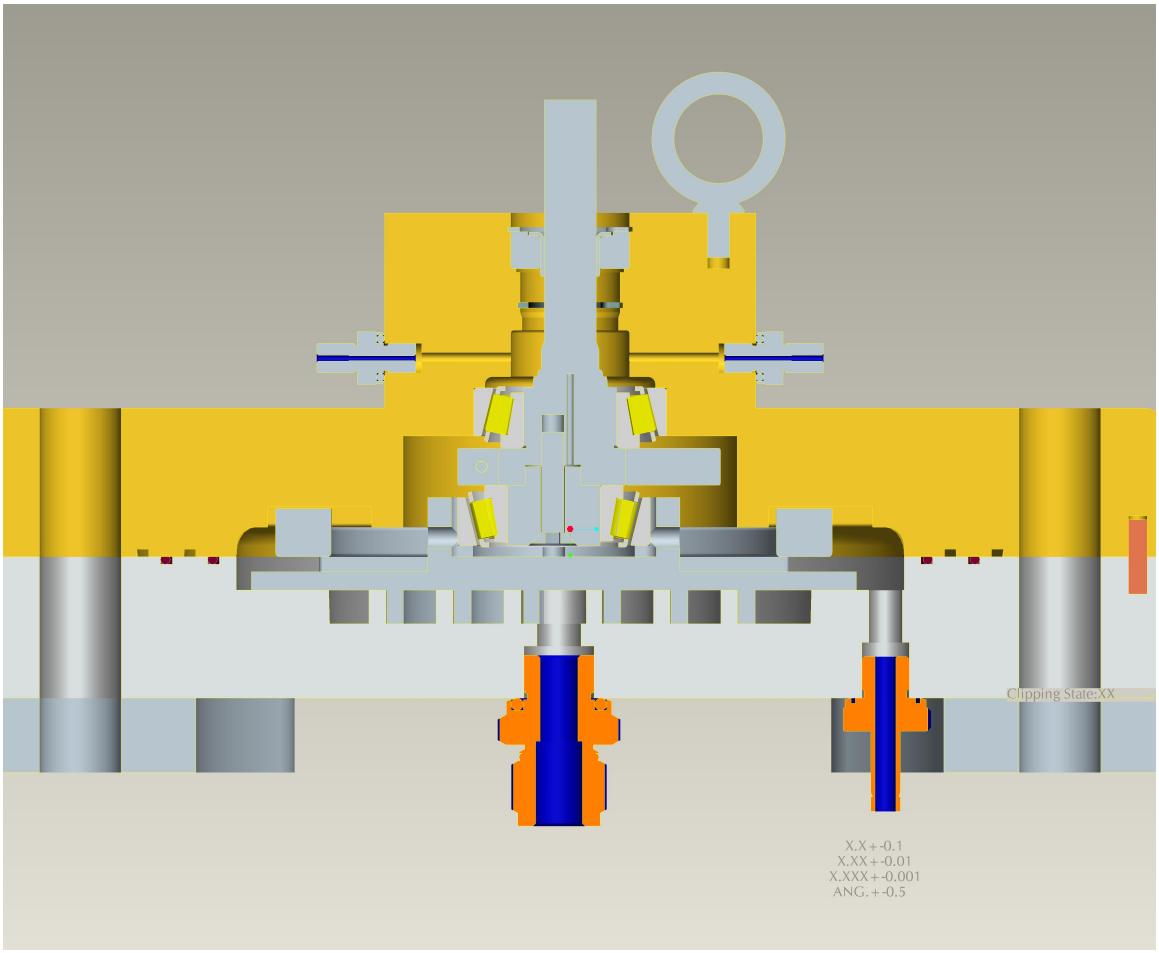


Figure G.1: Continued.

Outer Involute

$$
\begin{aligned}
x_{outer} &= r_b \left[\cos\phi + (\phi - \phi_{o0})\sin\phi\right] \\
y_{outer} &= r_b \left[\sin\phi - (\phi - \phi_{o0})\cos\phi\right] \\
\phi_{os} &< \phi < \phi_{oe}
\end{aligned}
$$

Curve A

$$
\begin{aligned}
x &= x_a + r_a \cos t \\
y &= y_a + r_a \sin t \\
t_{1a} &< t < t_{2a}
\end{aligned}
$$

Curve B

$$
\begin{aligned}
x &= x_b + r_b \cos t \\
y &= y_b + r_b \sin t \\
t_{1b} &< t < t_{2b}
\end{aligned}
$$

Figure G.1: Continued.

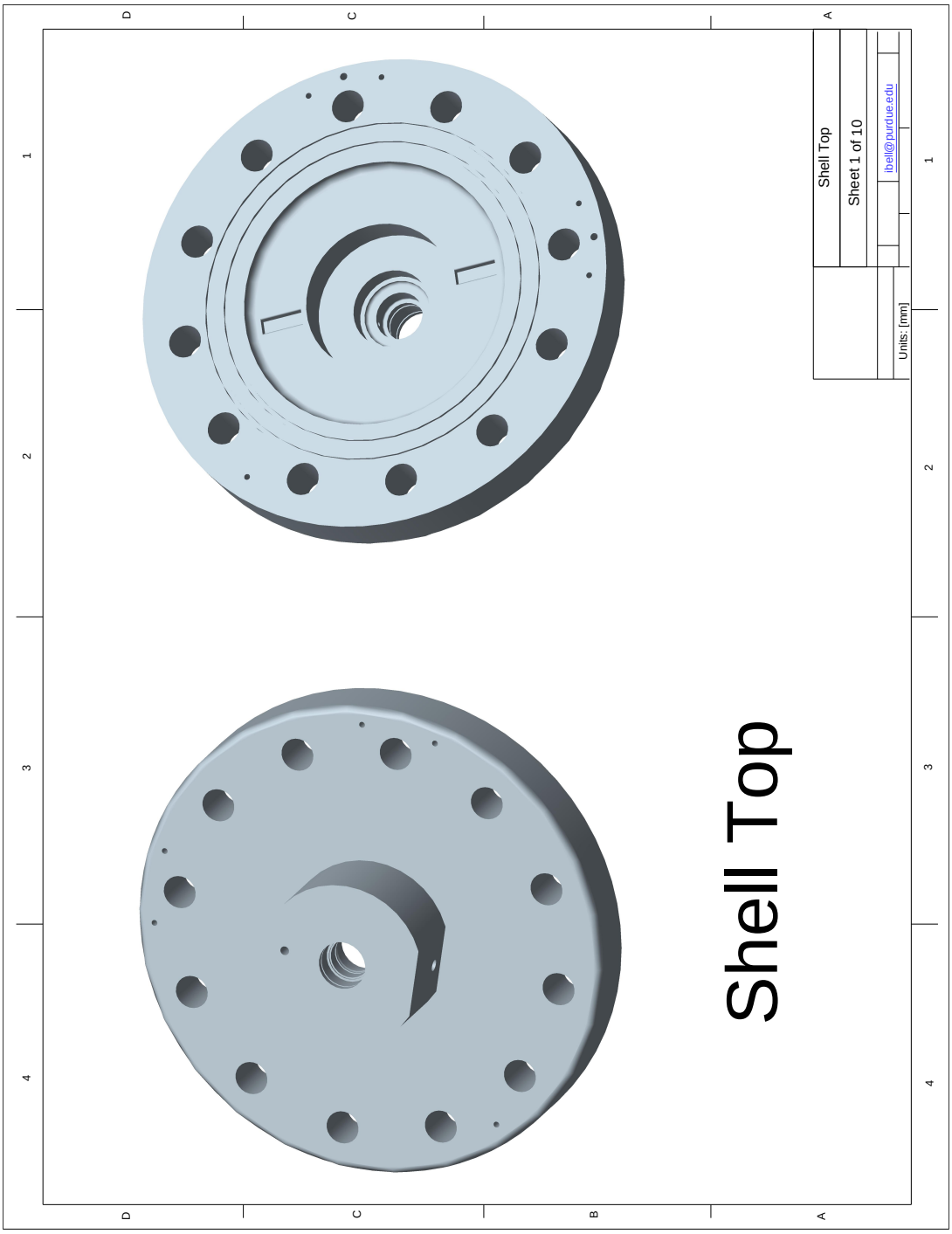


Figure G.1: Continued.

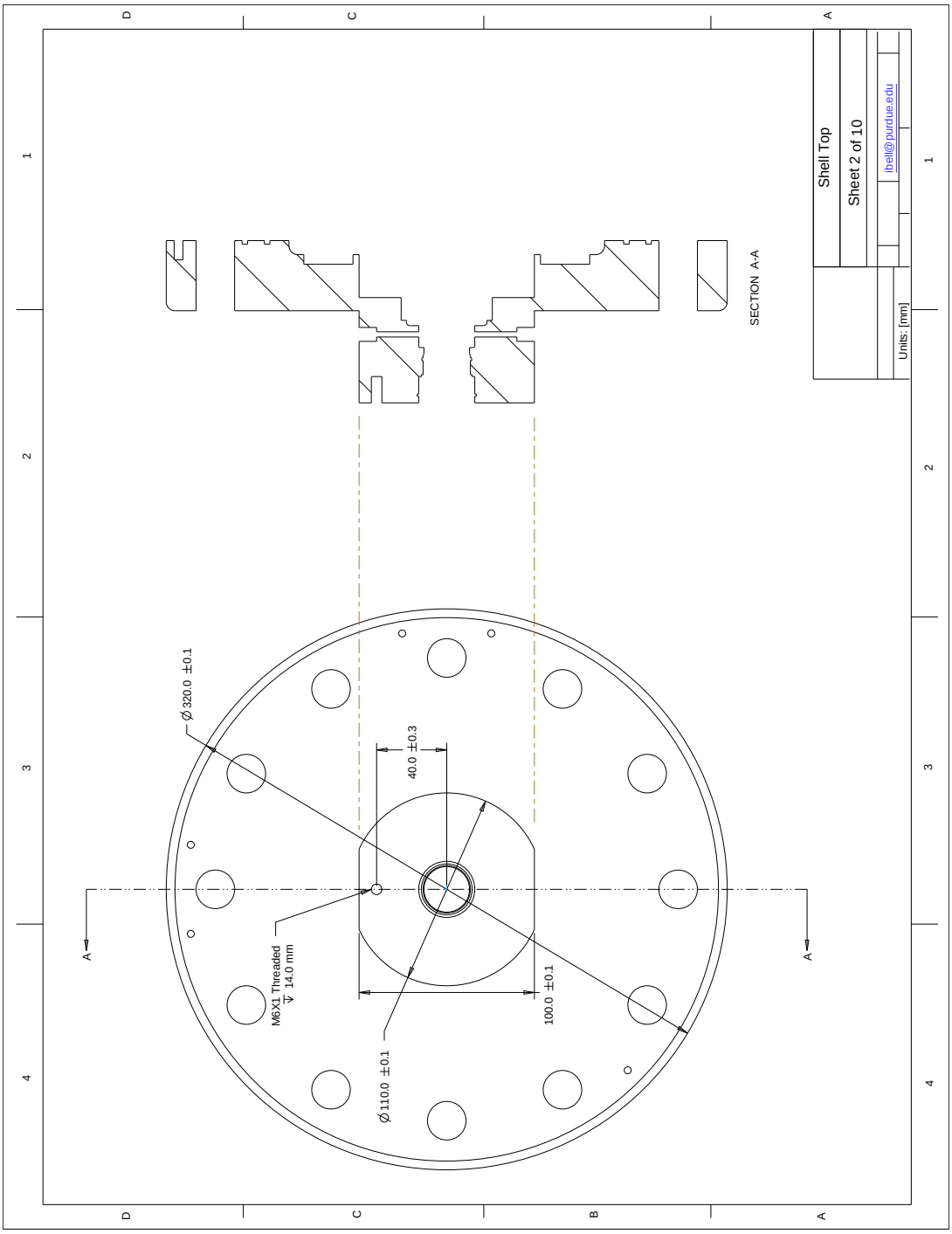


Figure G.1: Continued.

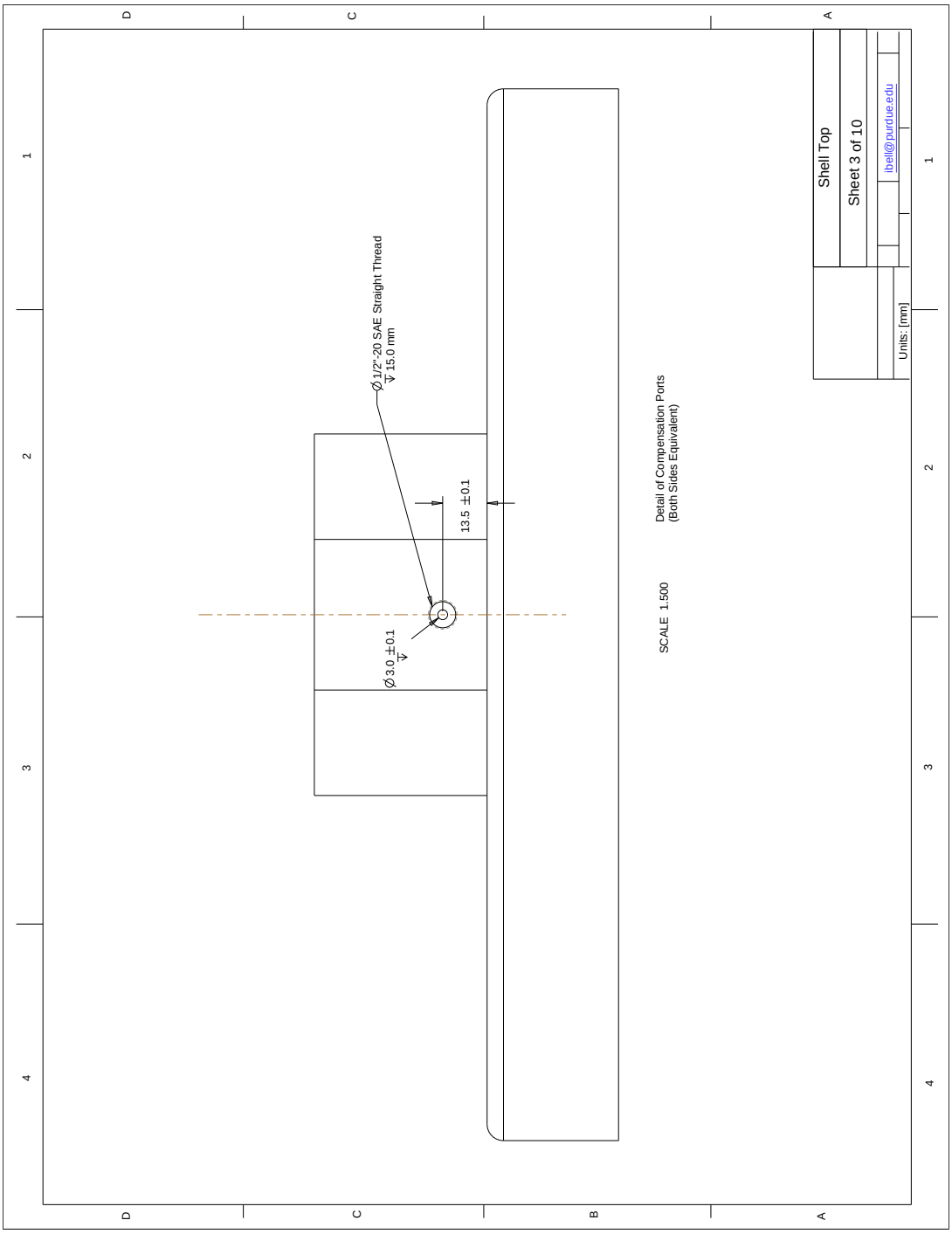


Figure G.1: Continued.

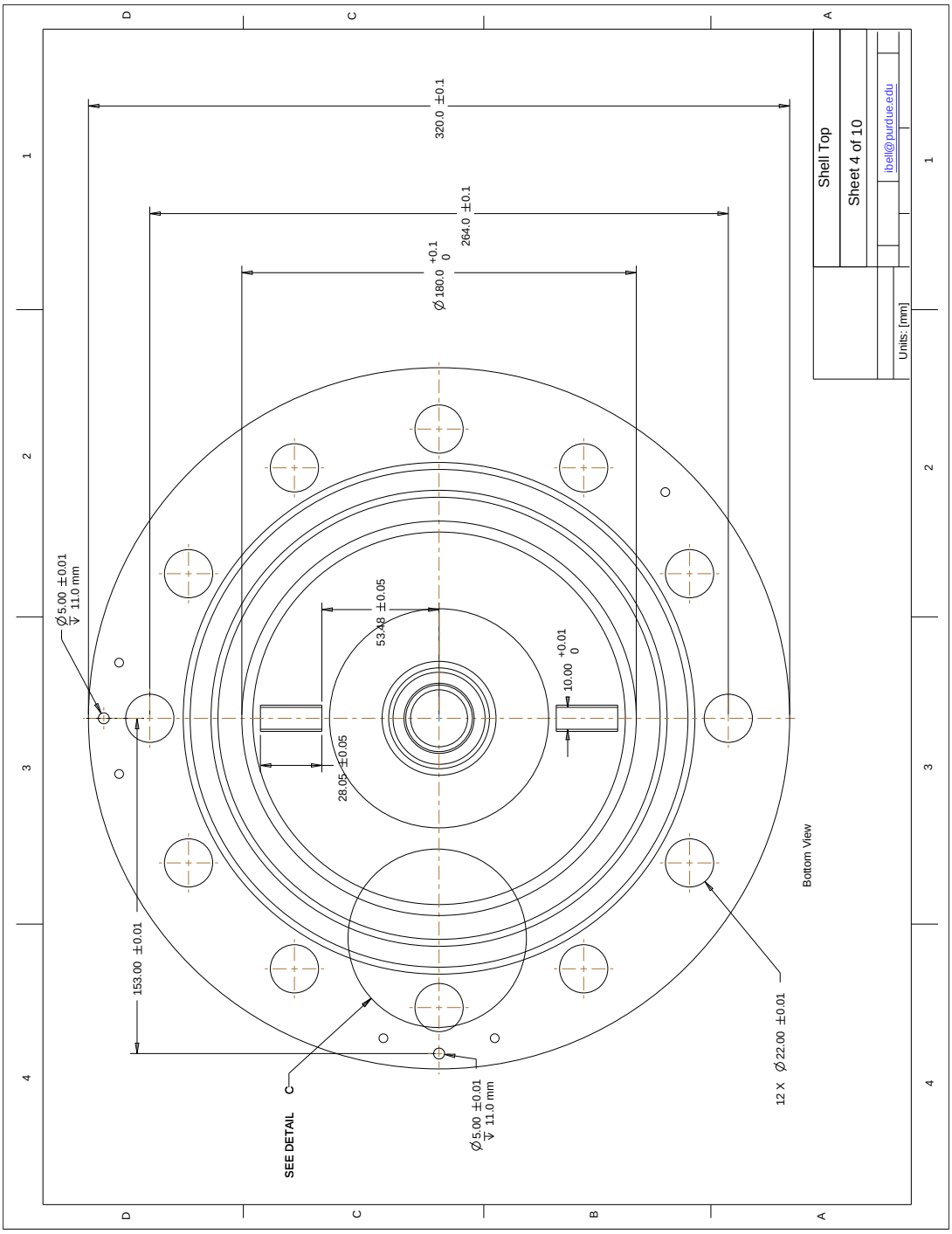


Figure G.1: Continued.

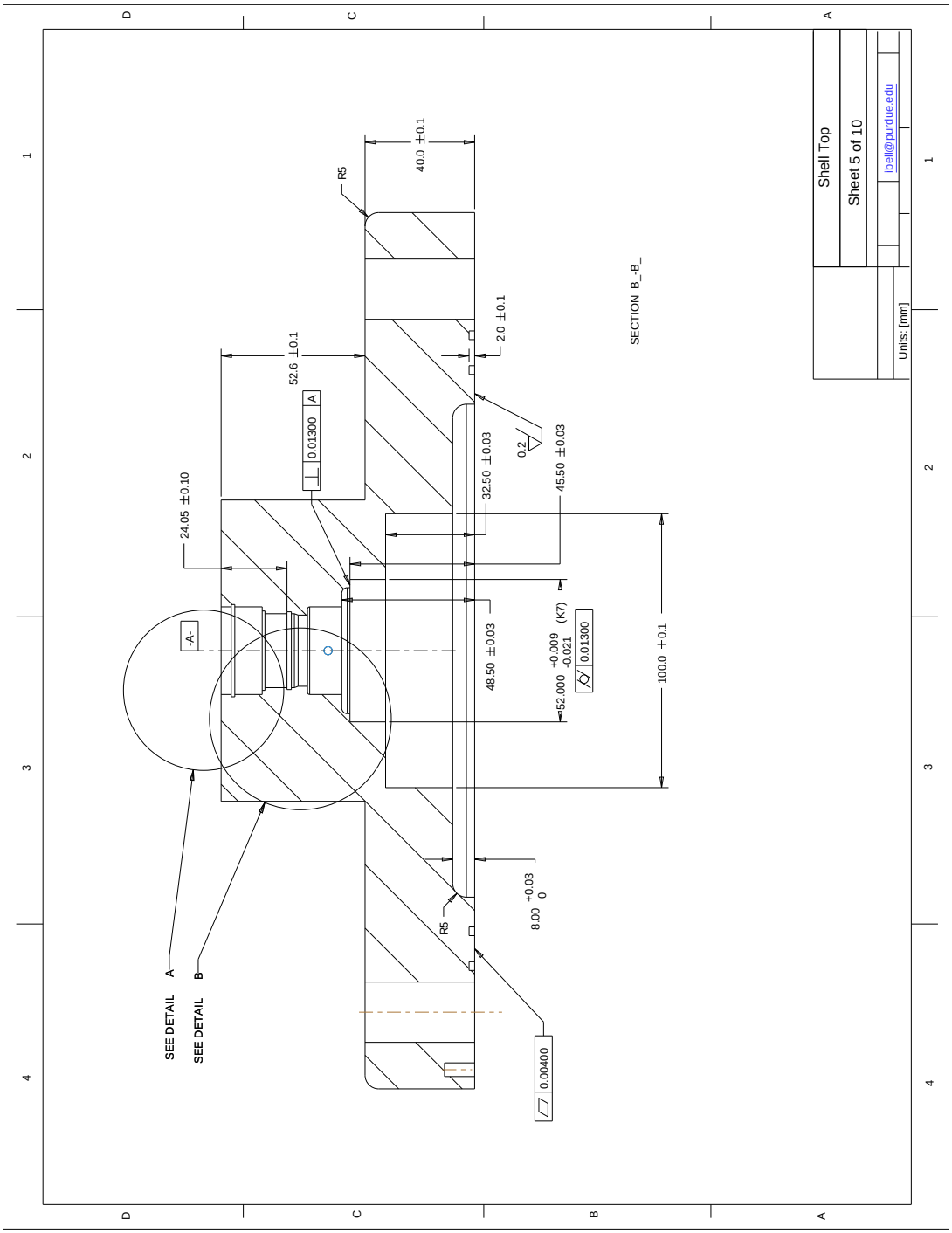


Figure G.1: Continued.

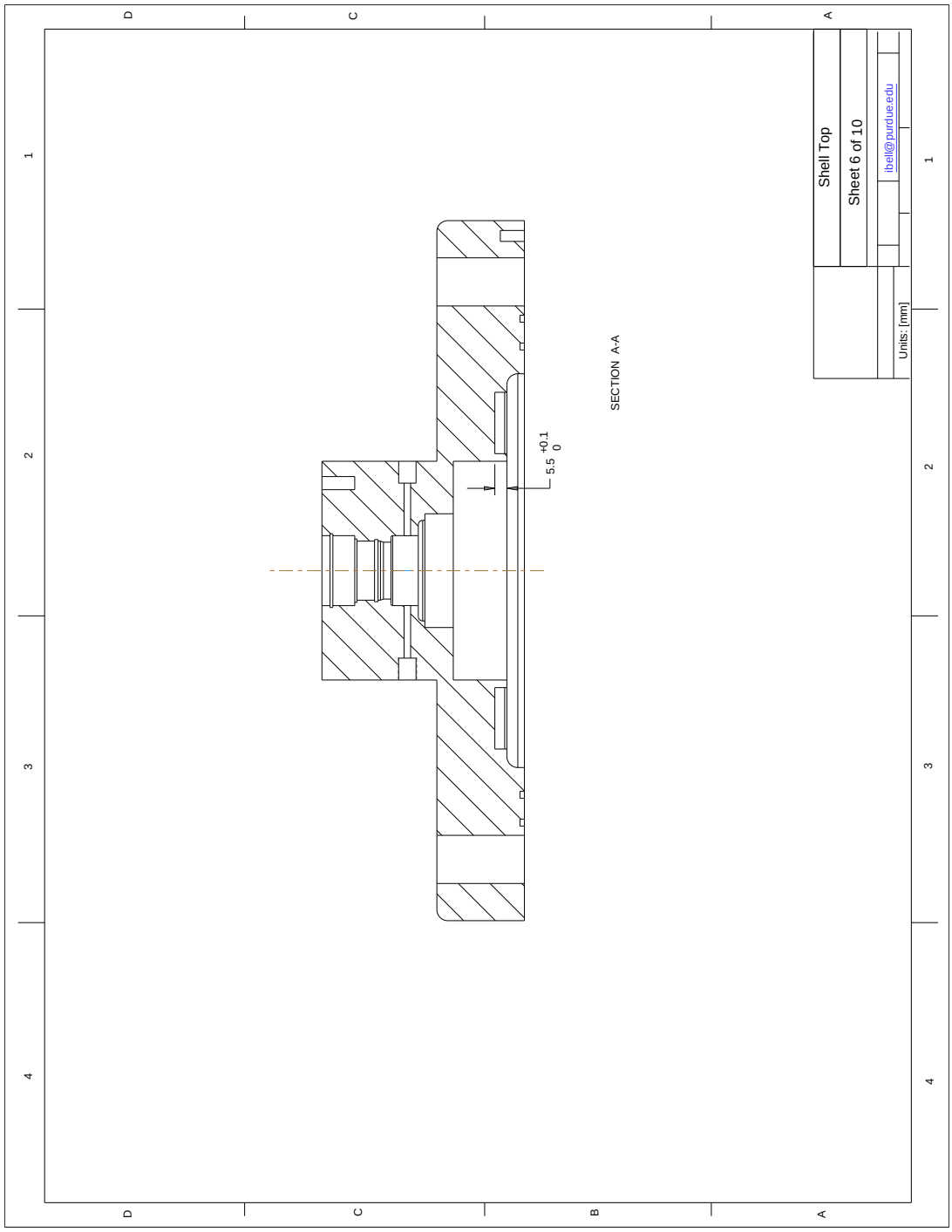


Figure G.1: Continued.

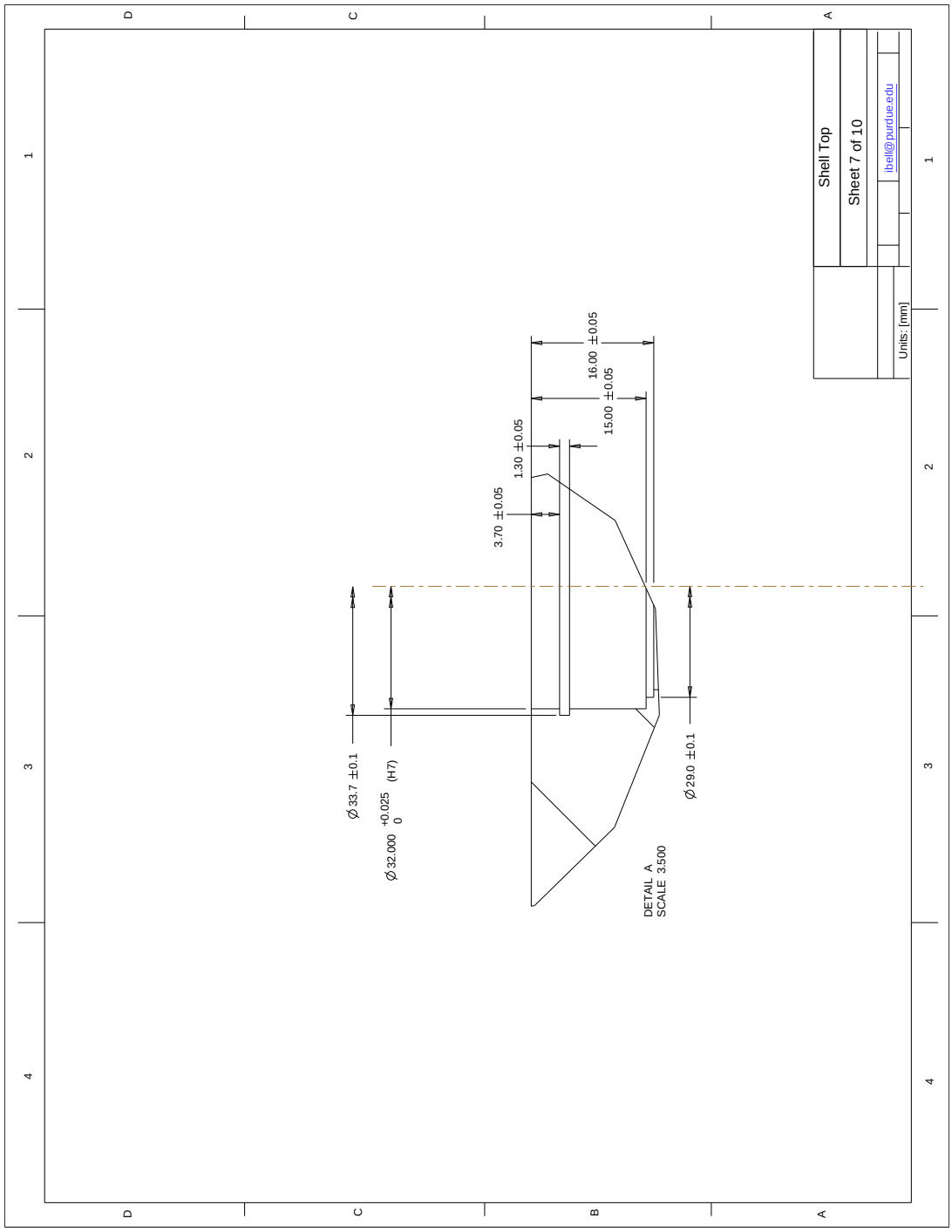


Figure G.1: Continued.

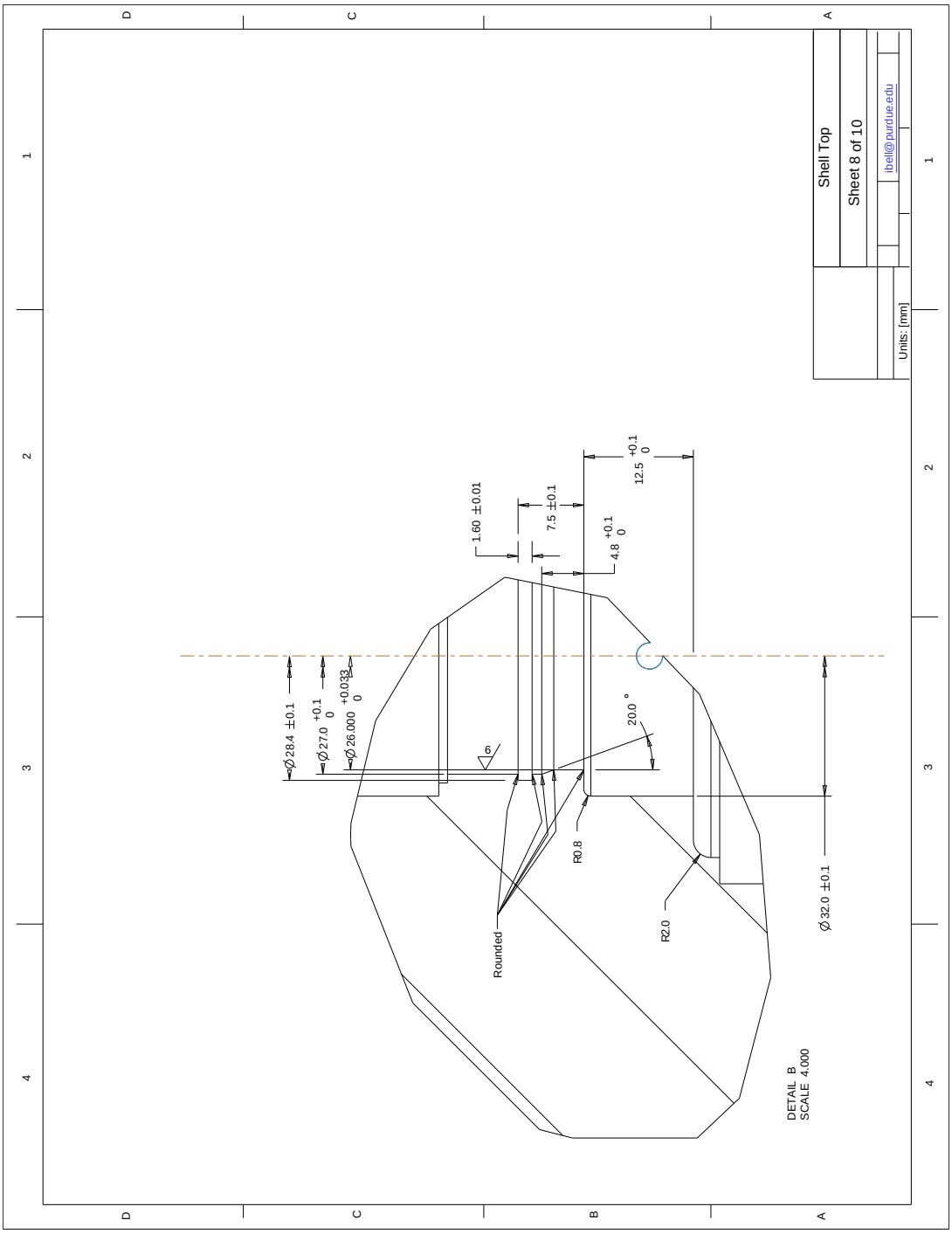


Figure G.1: Continued.

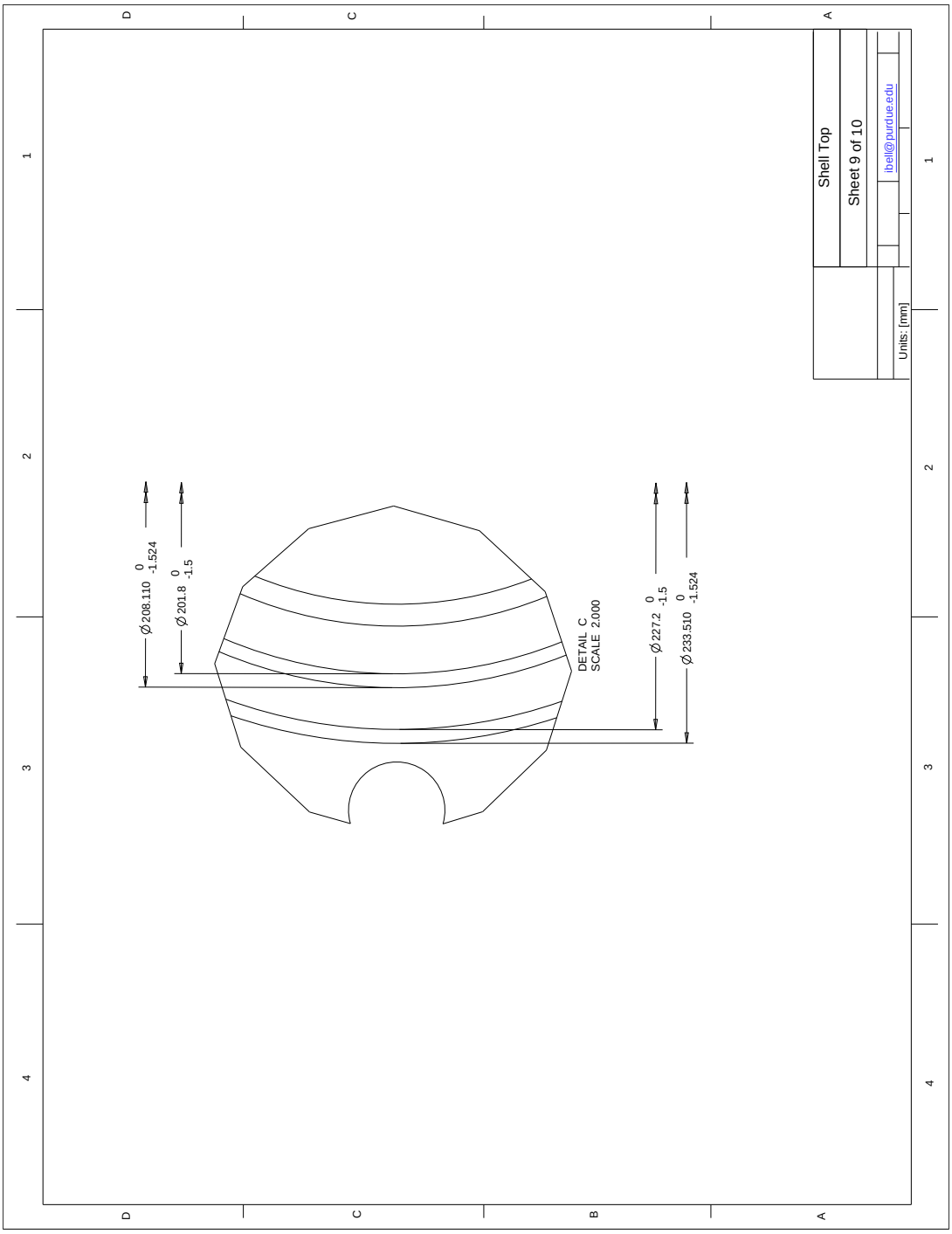


Figure G.1: Continued.

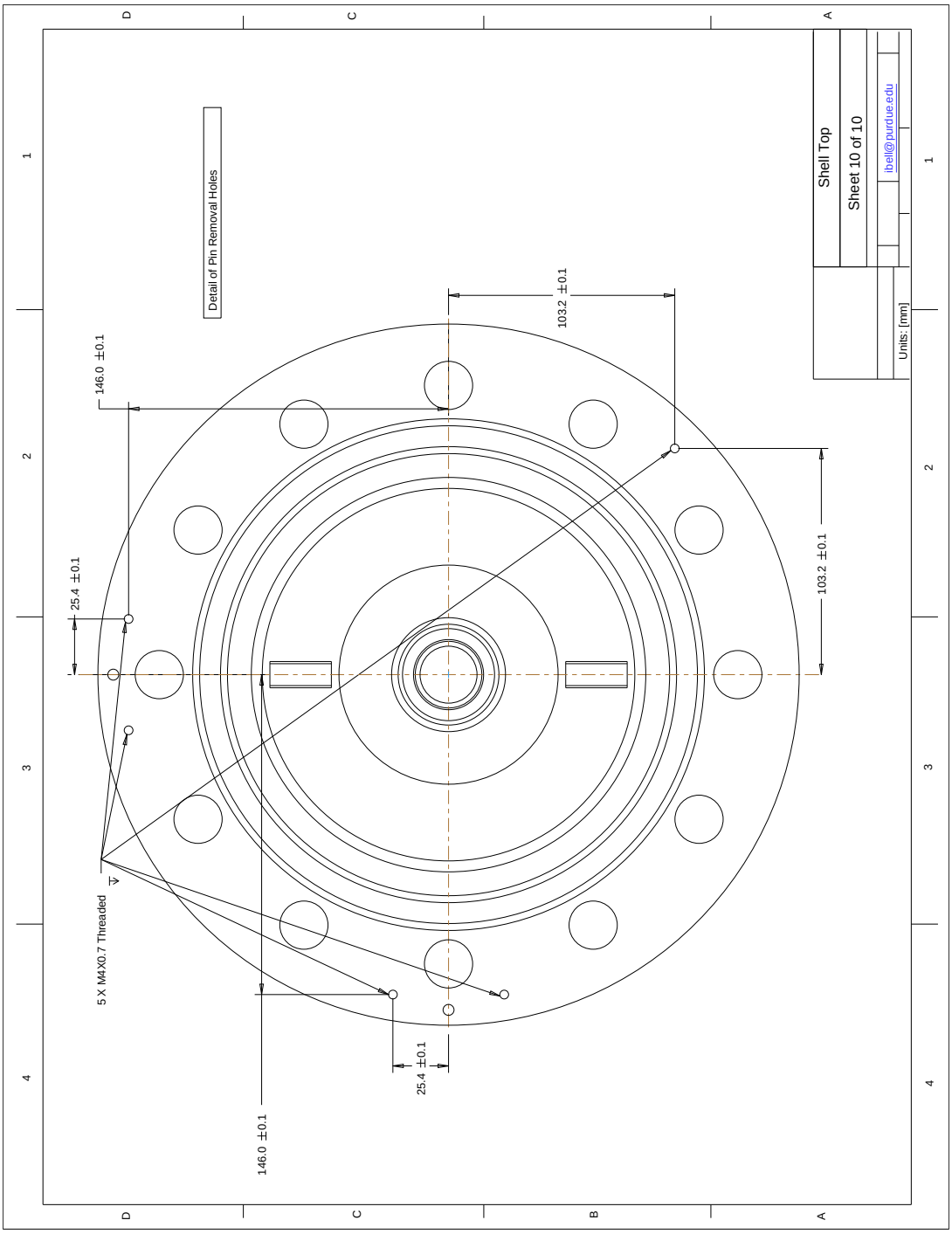


Figure G.1: Continued.

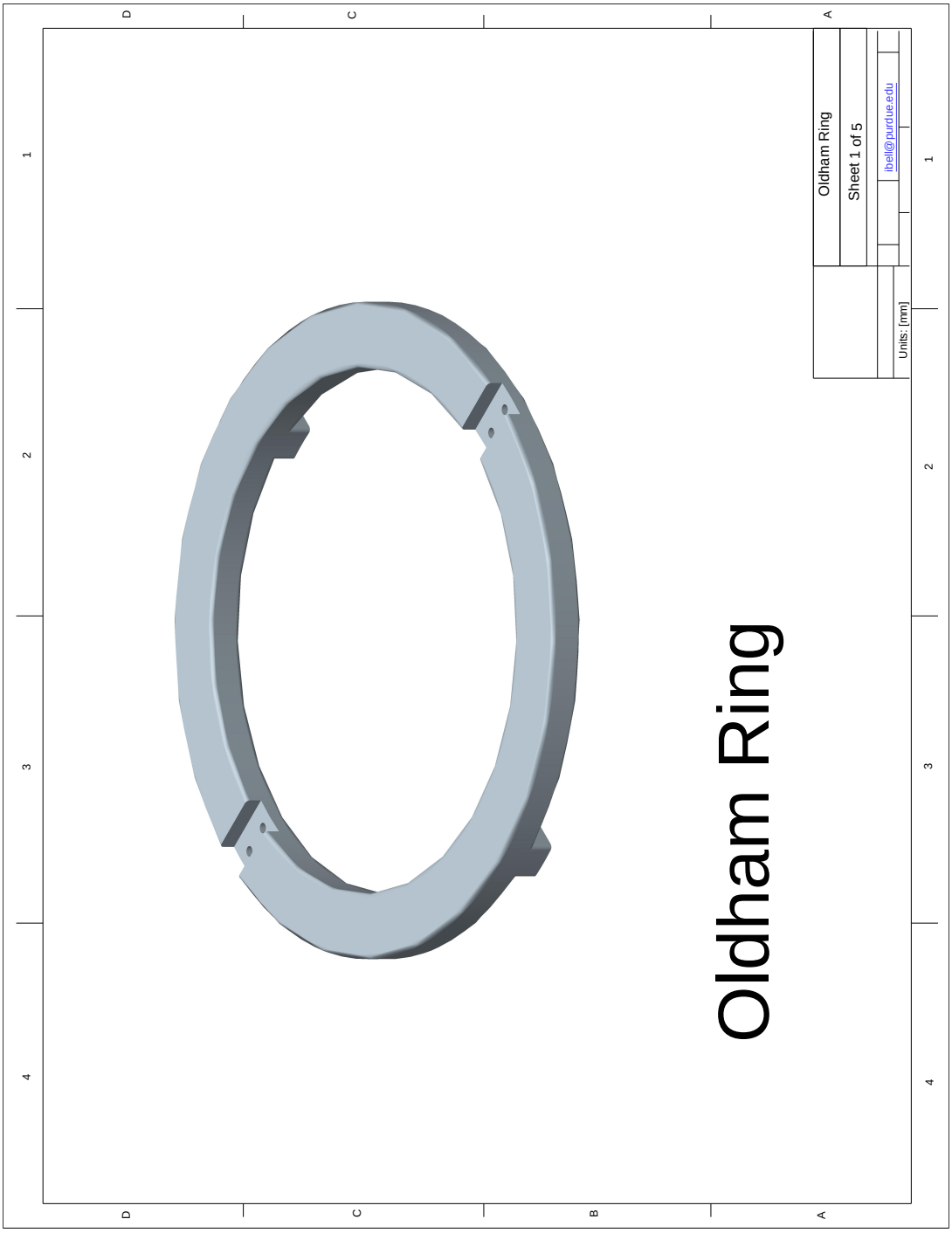


Figure G.1: Continued.

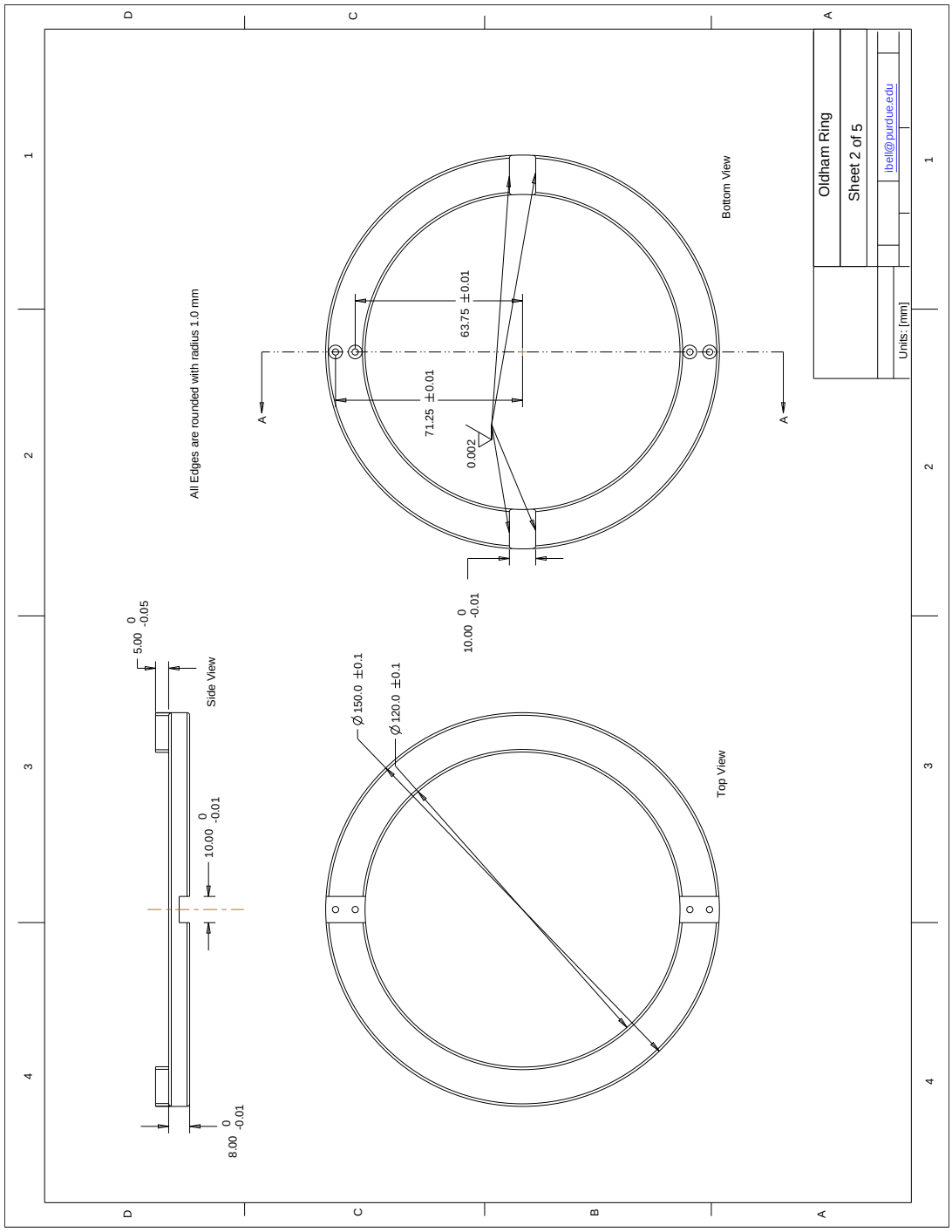


Figure G.1: Continued.

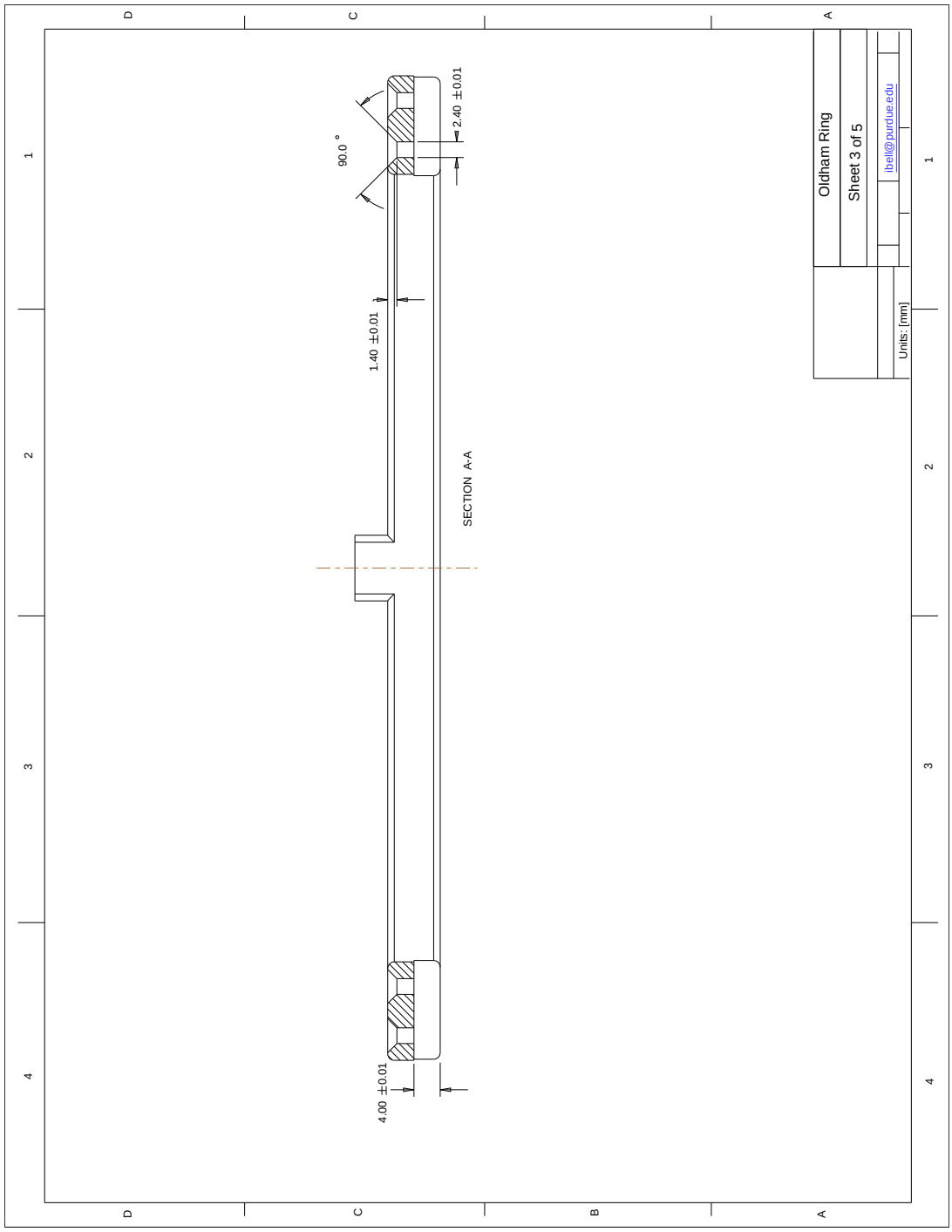


Figure G.1: Continued.

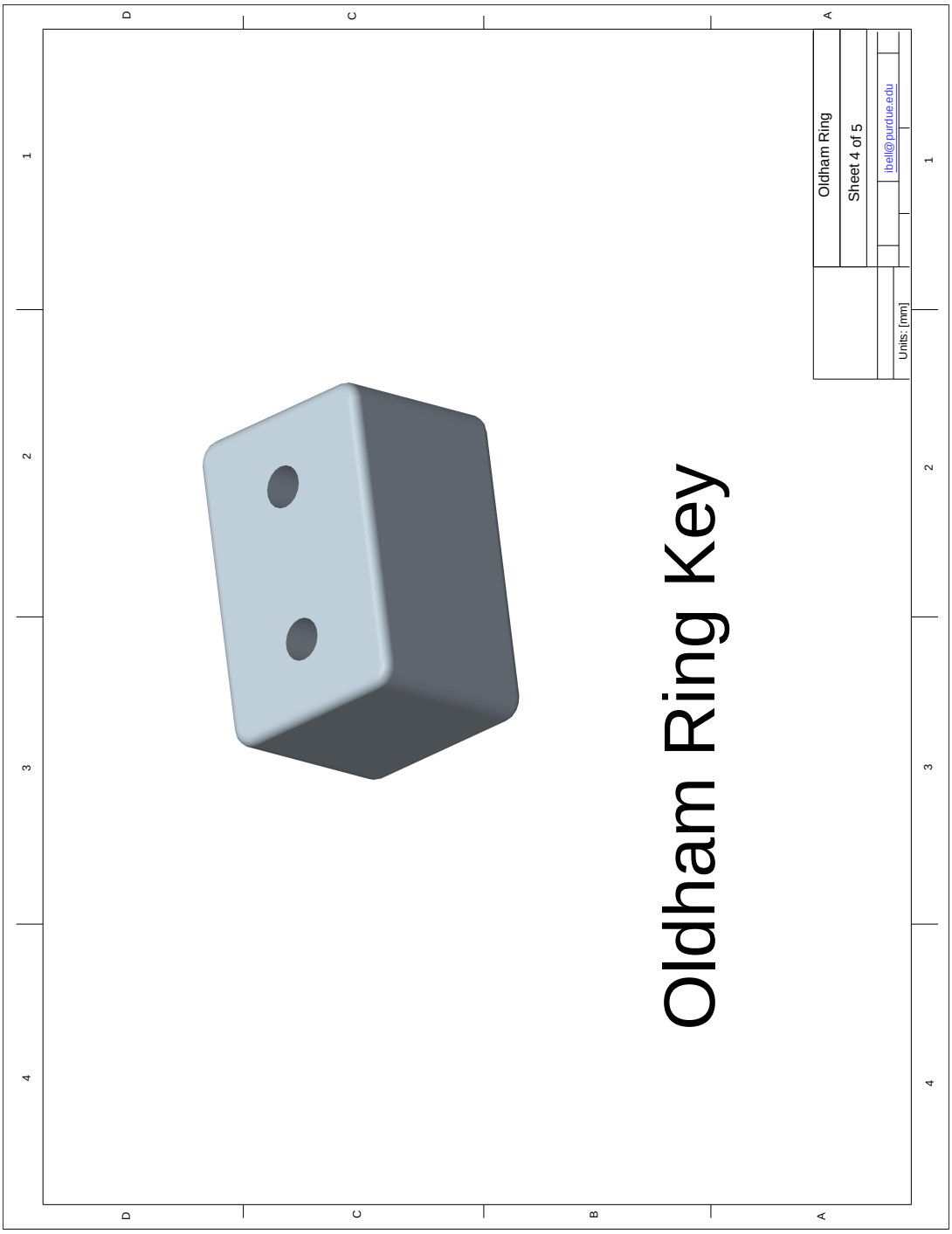


Figure G.1: Continued.

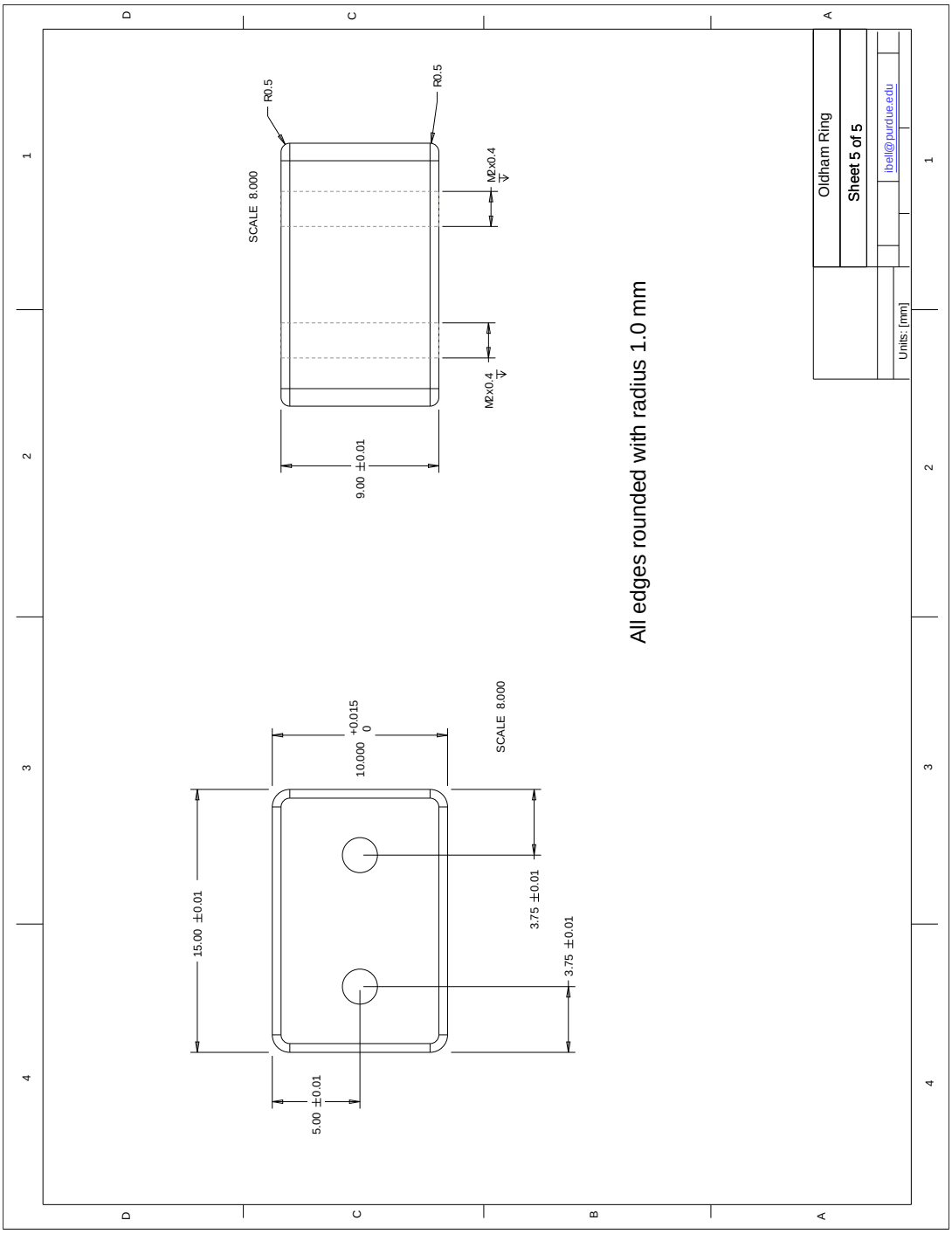


Figure G.1: Continued.

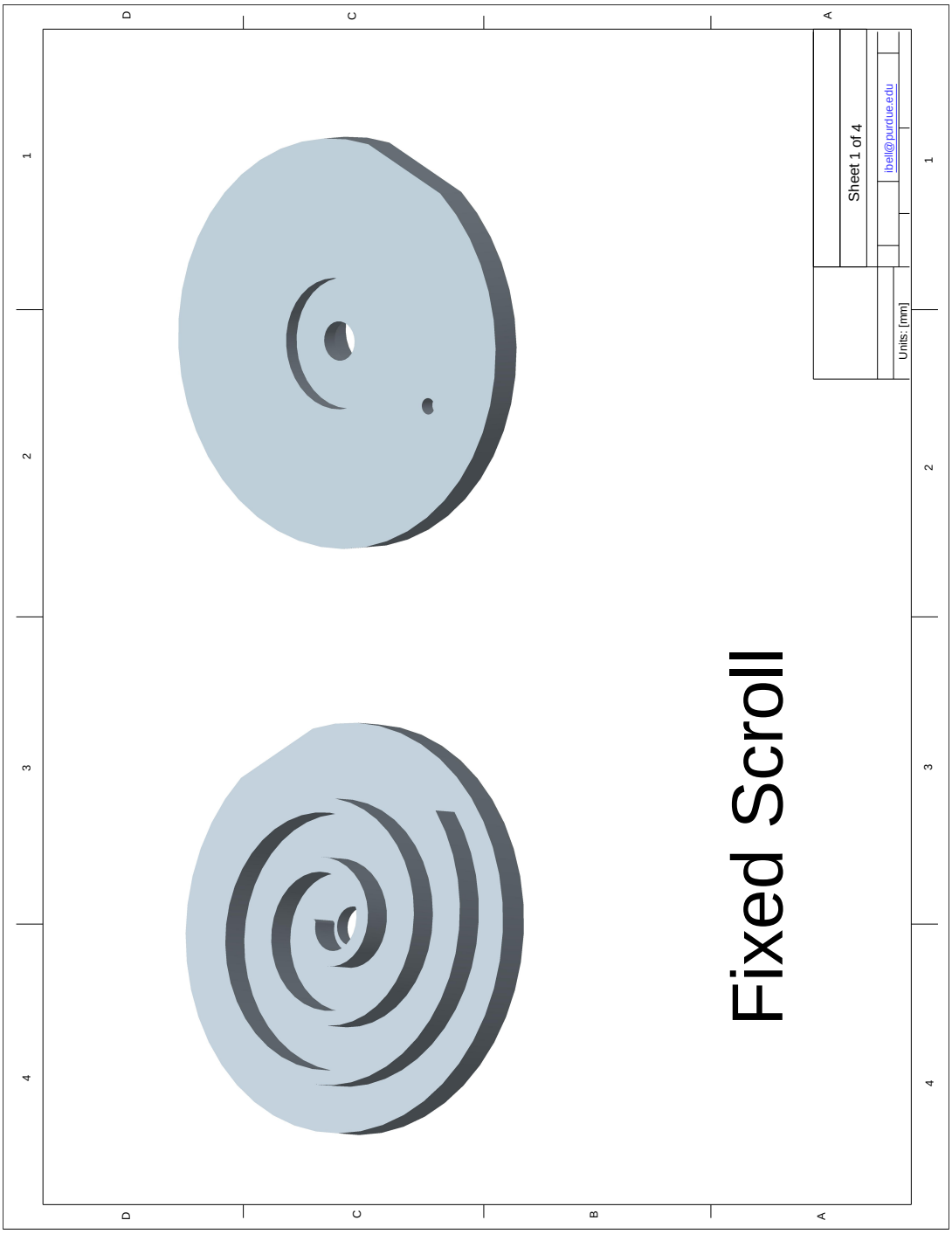


Figure G.1: Continued.

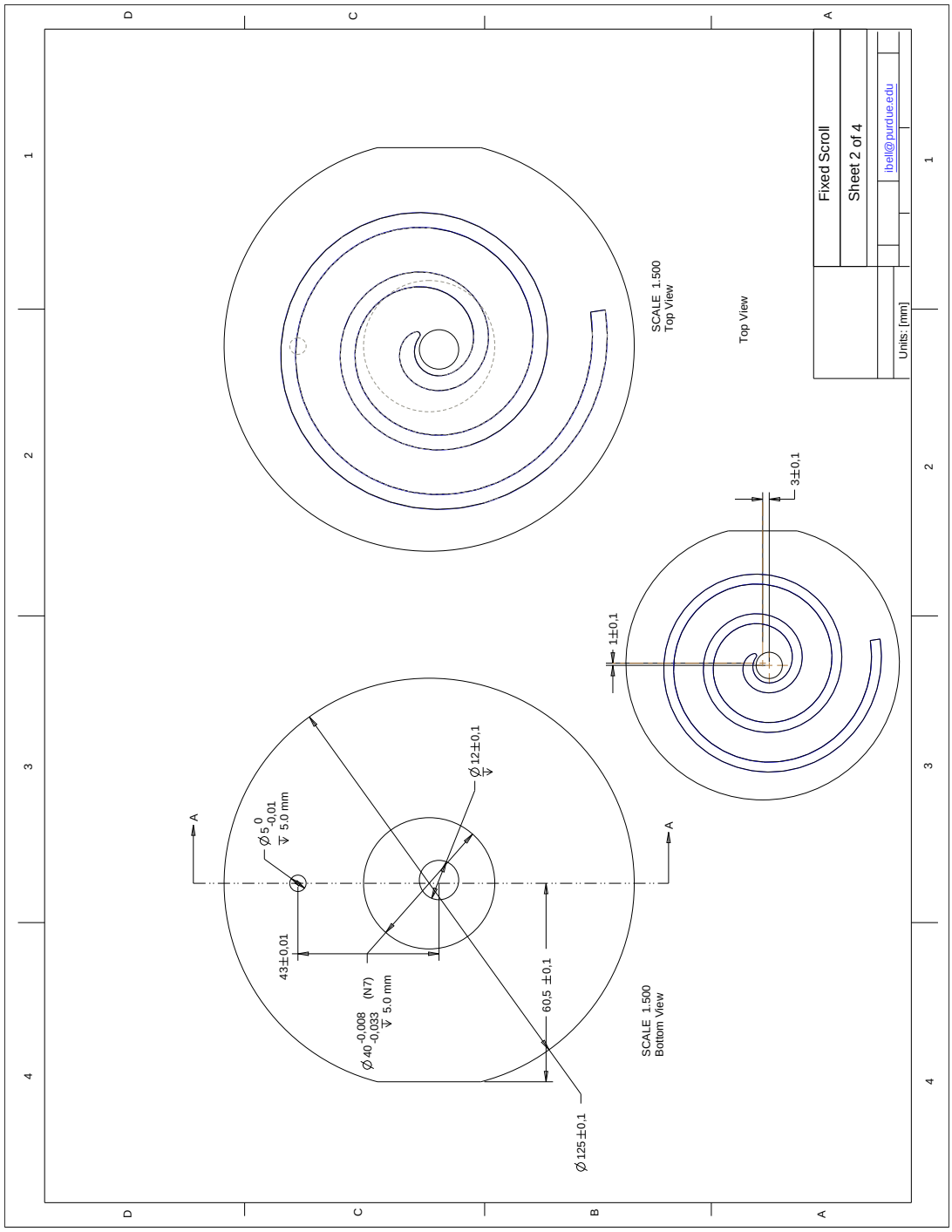


Figure G.1: Continued.

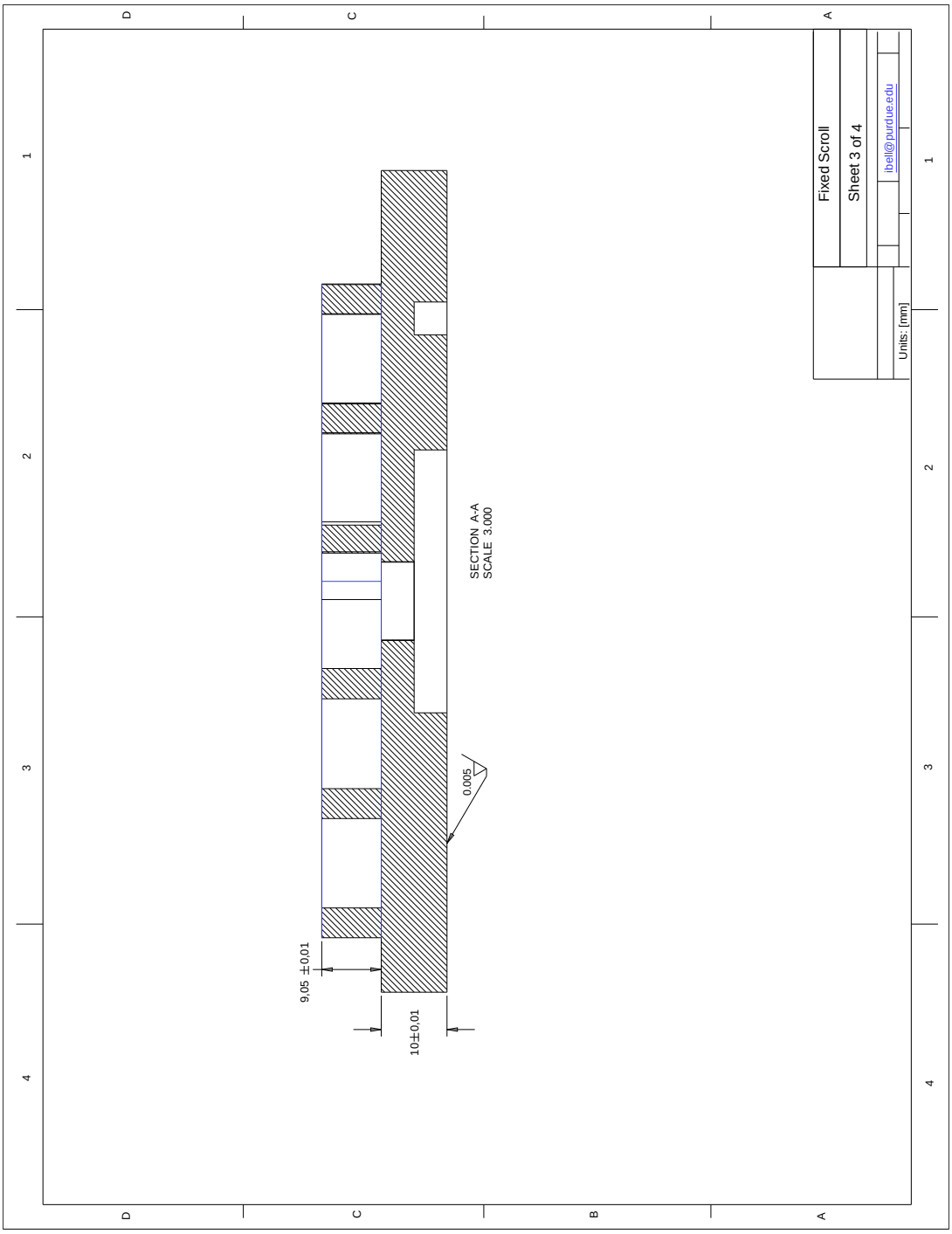


Figure G.1: Continued.

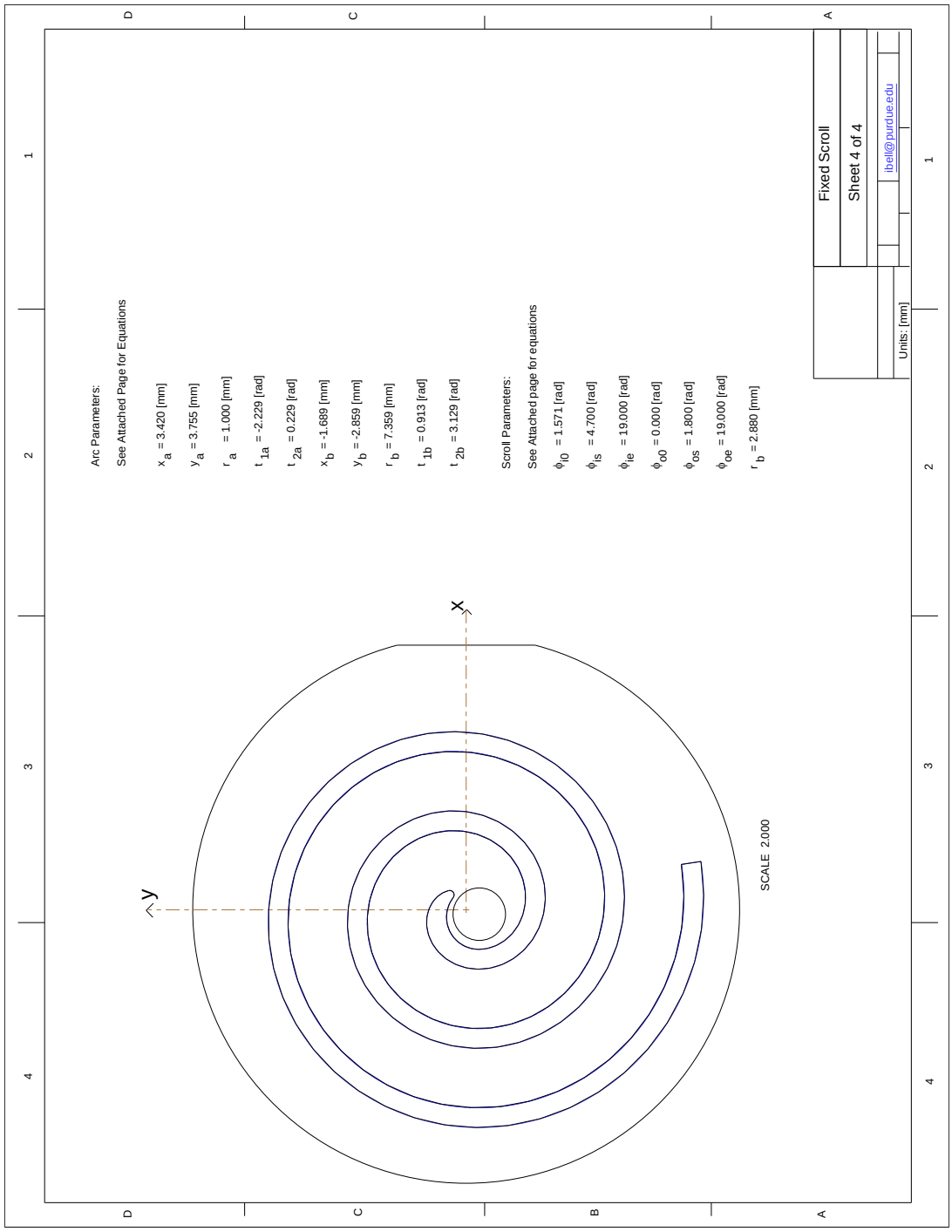


Figure G.1: Continued.

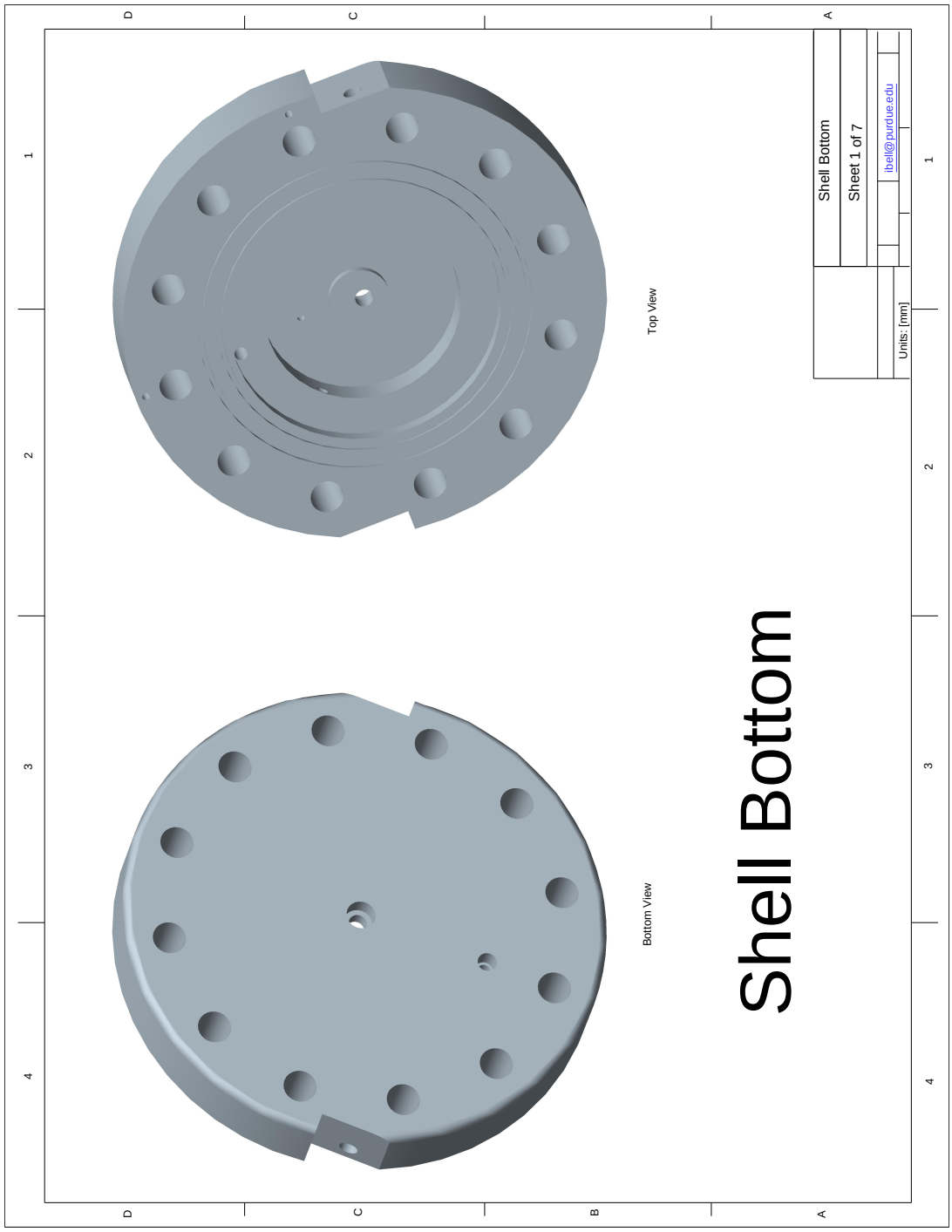


Figure G.1: Continued.

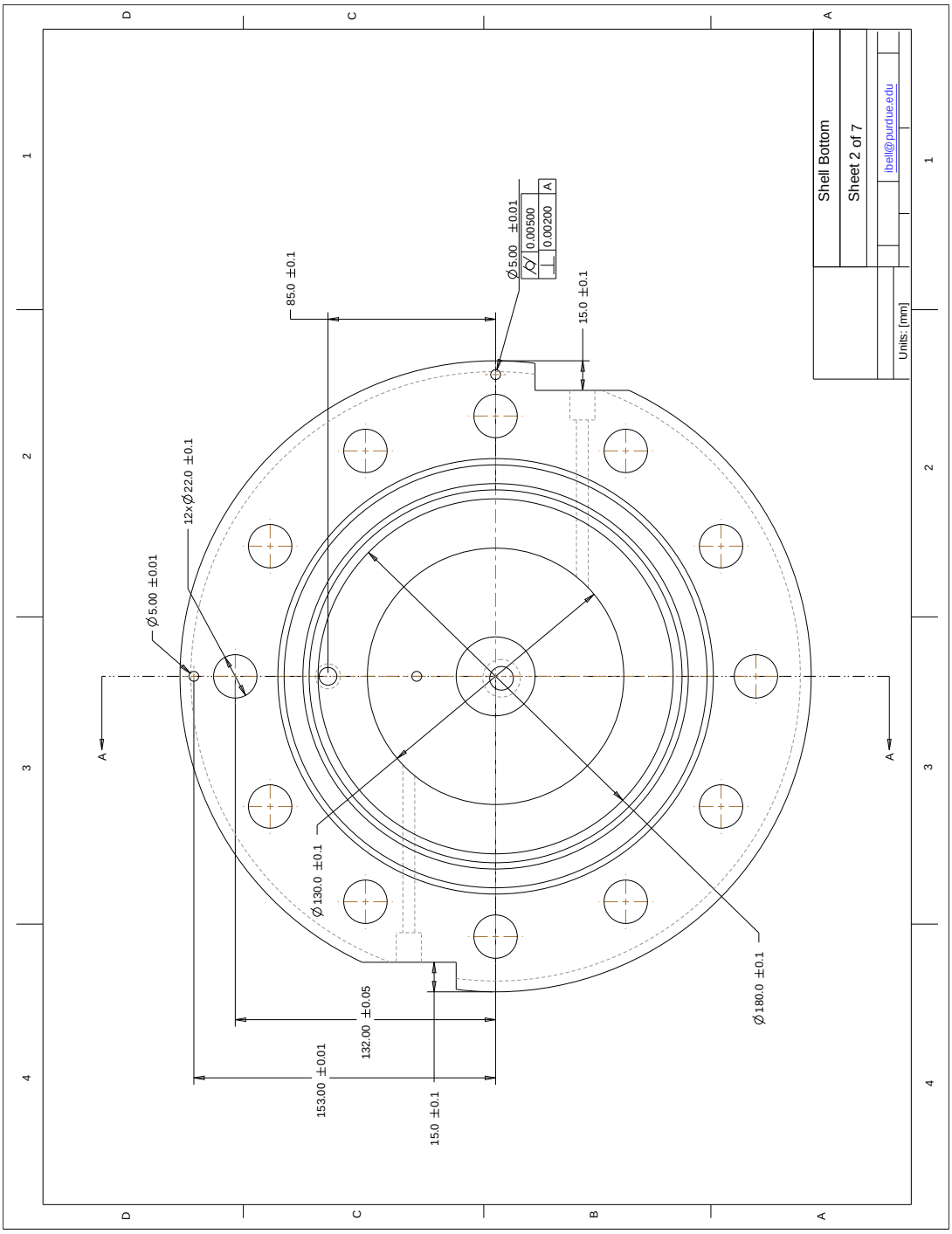


Figure G.1: Continued.

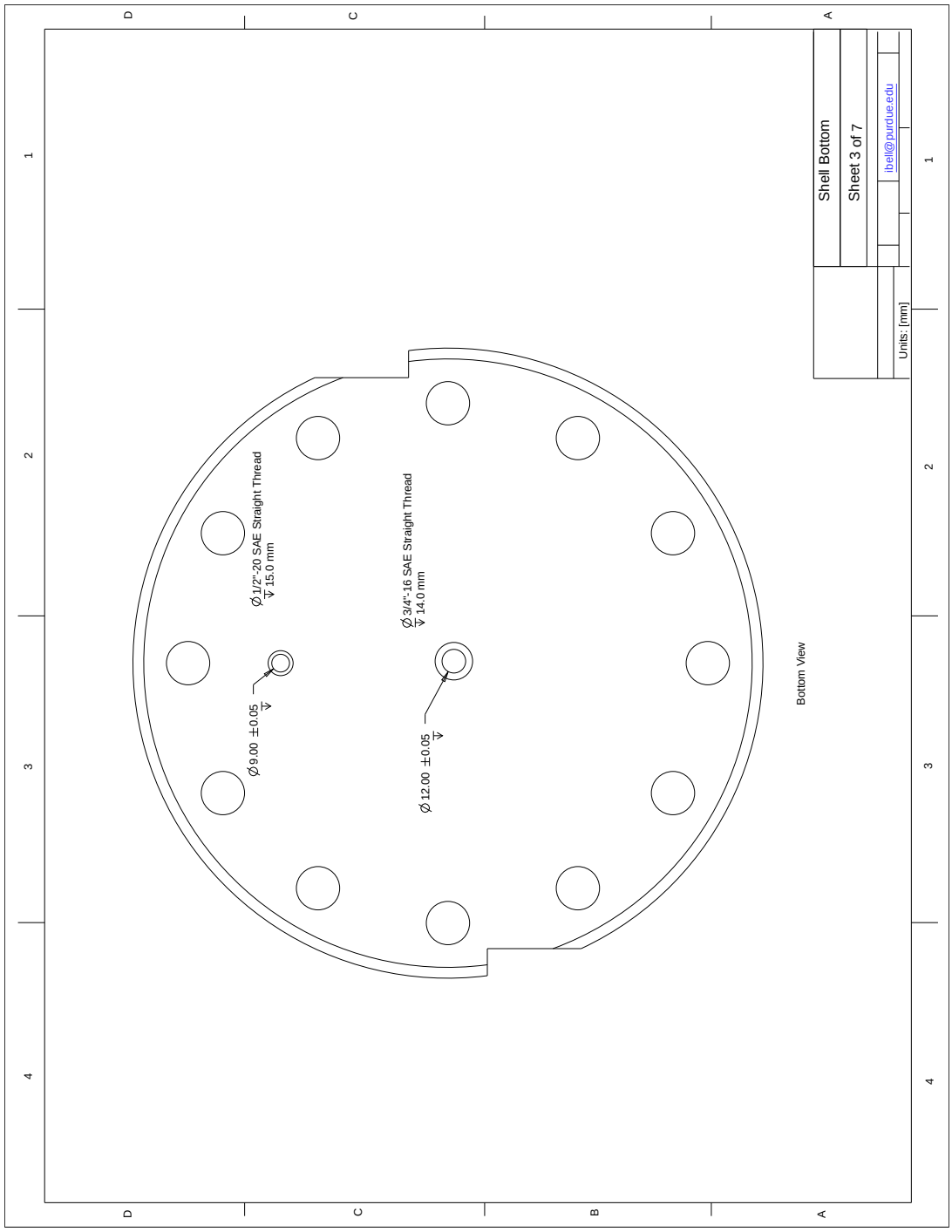


Figure G.1: Continued.

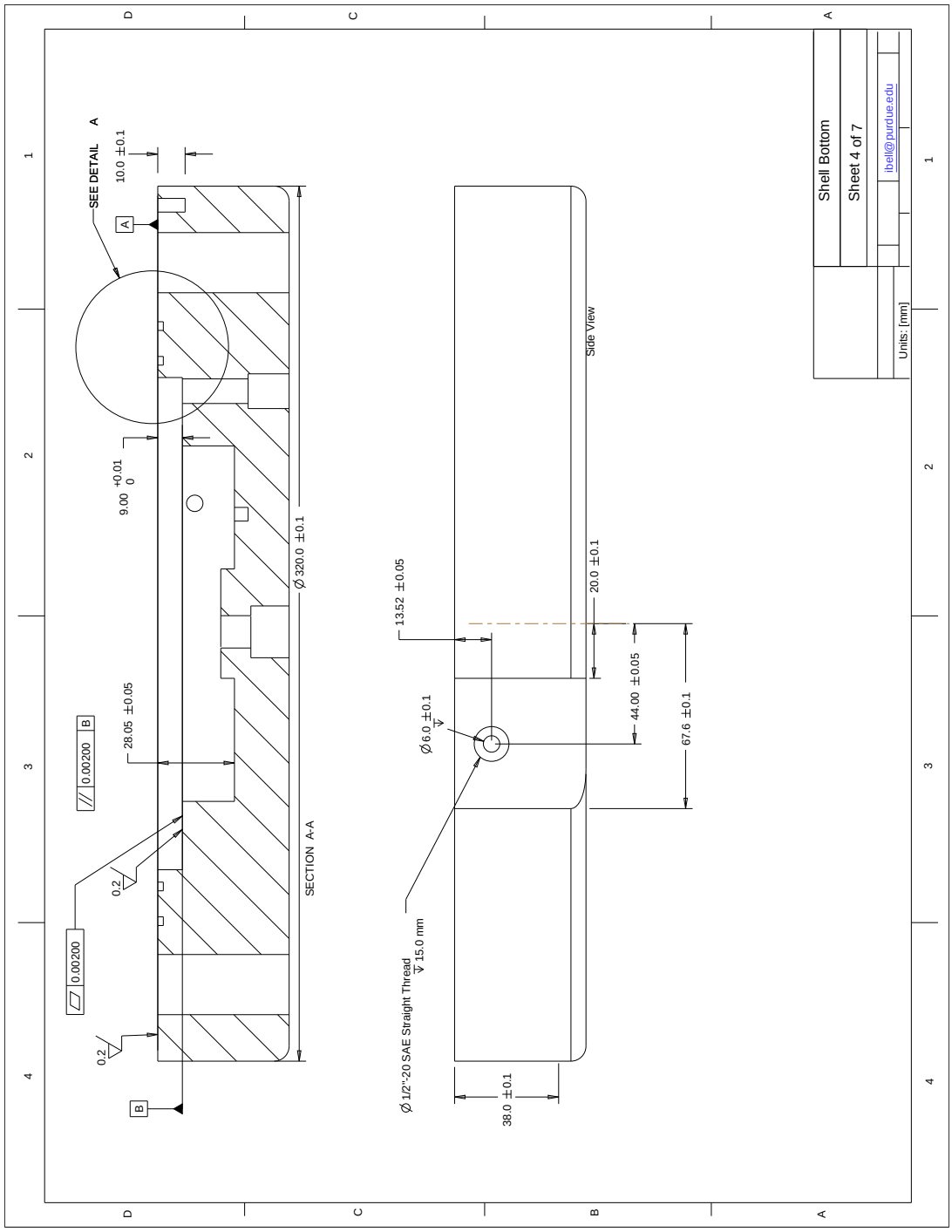


Figure G.1: Continued.

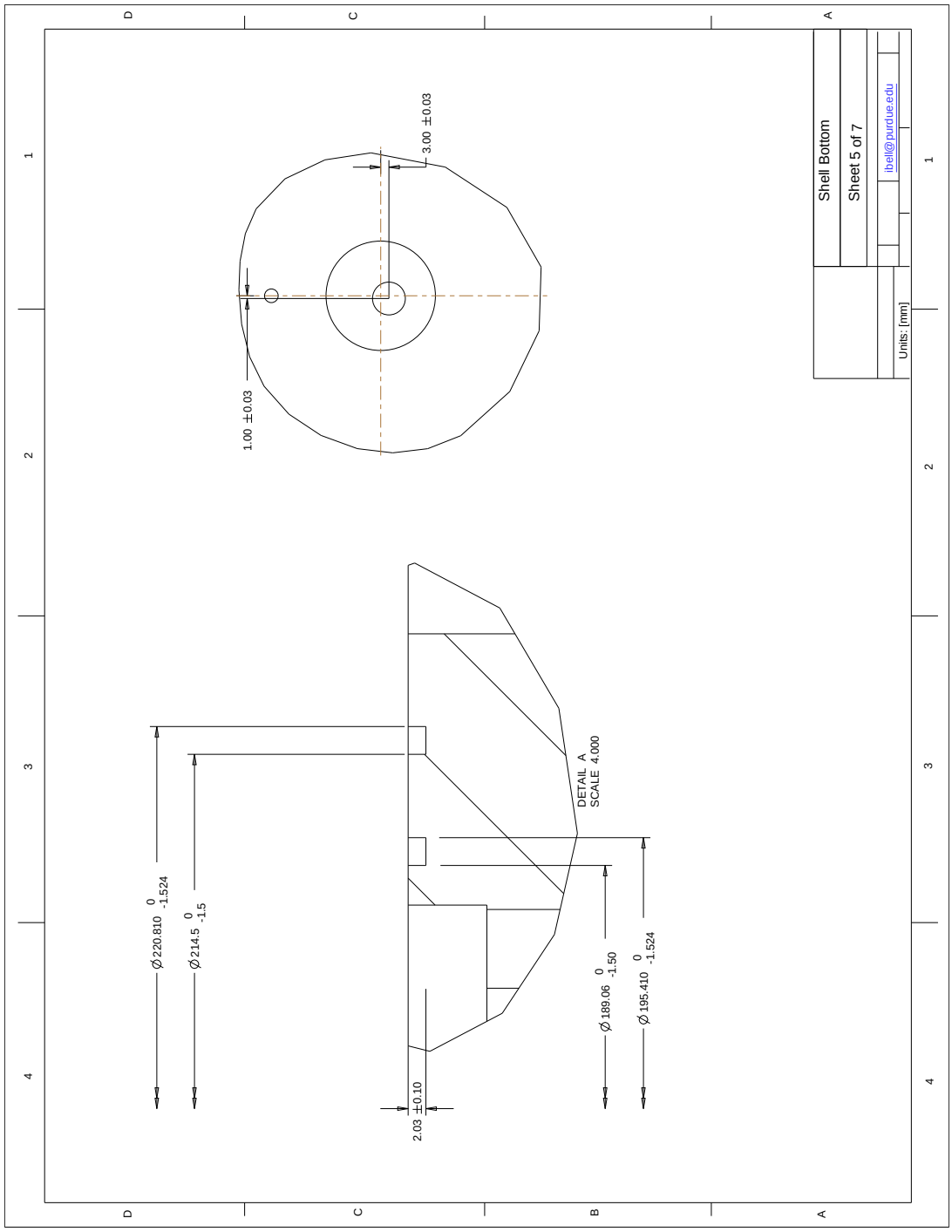


Figure G.1: Continued.

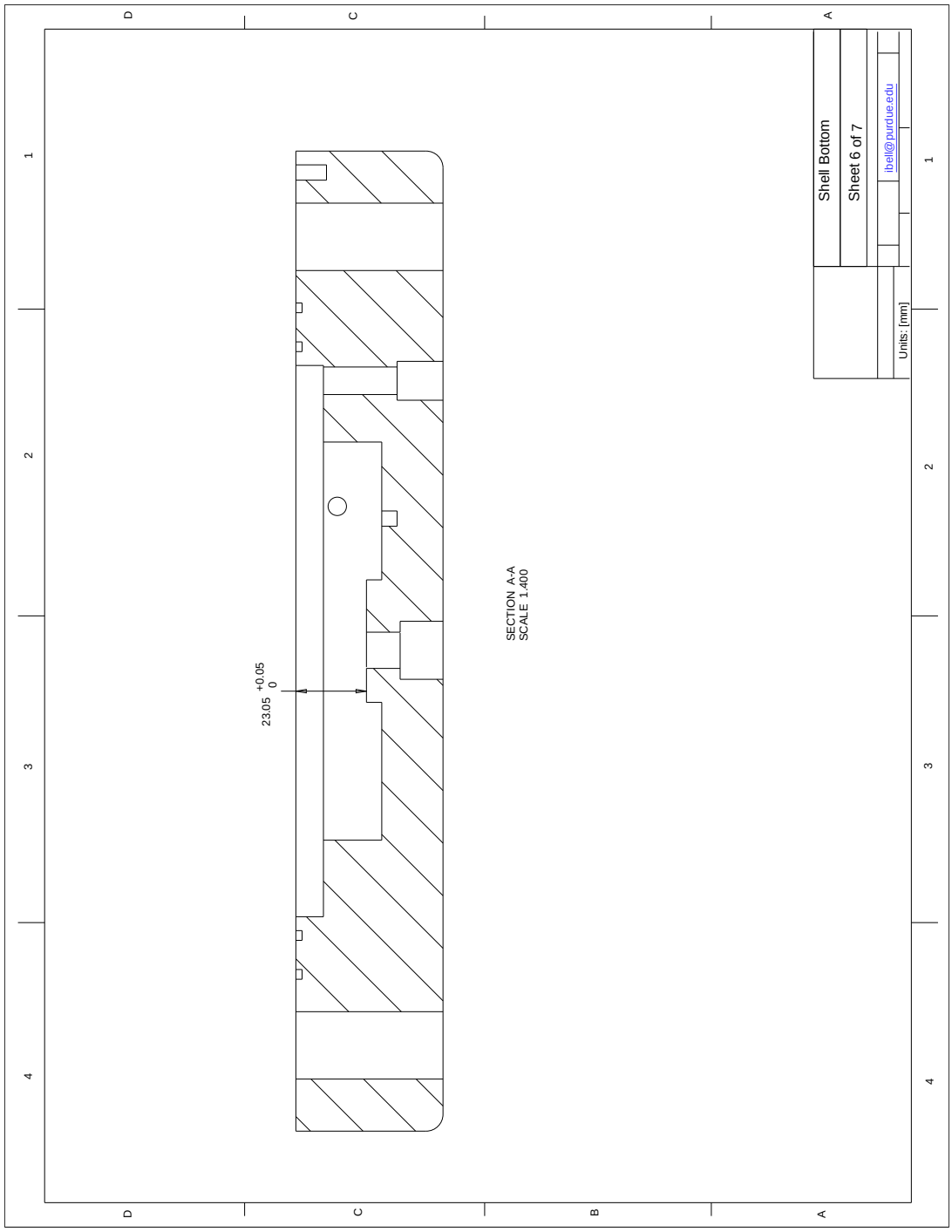


Figure G.1: Continued.

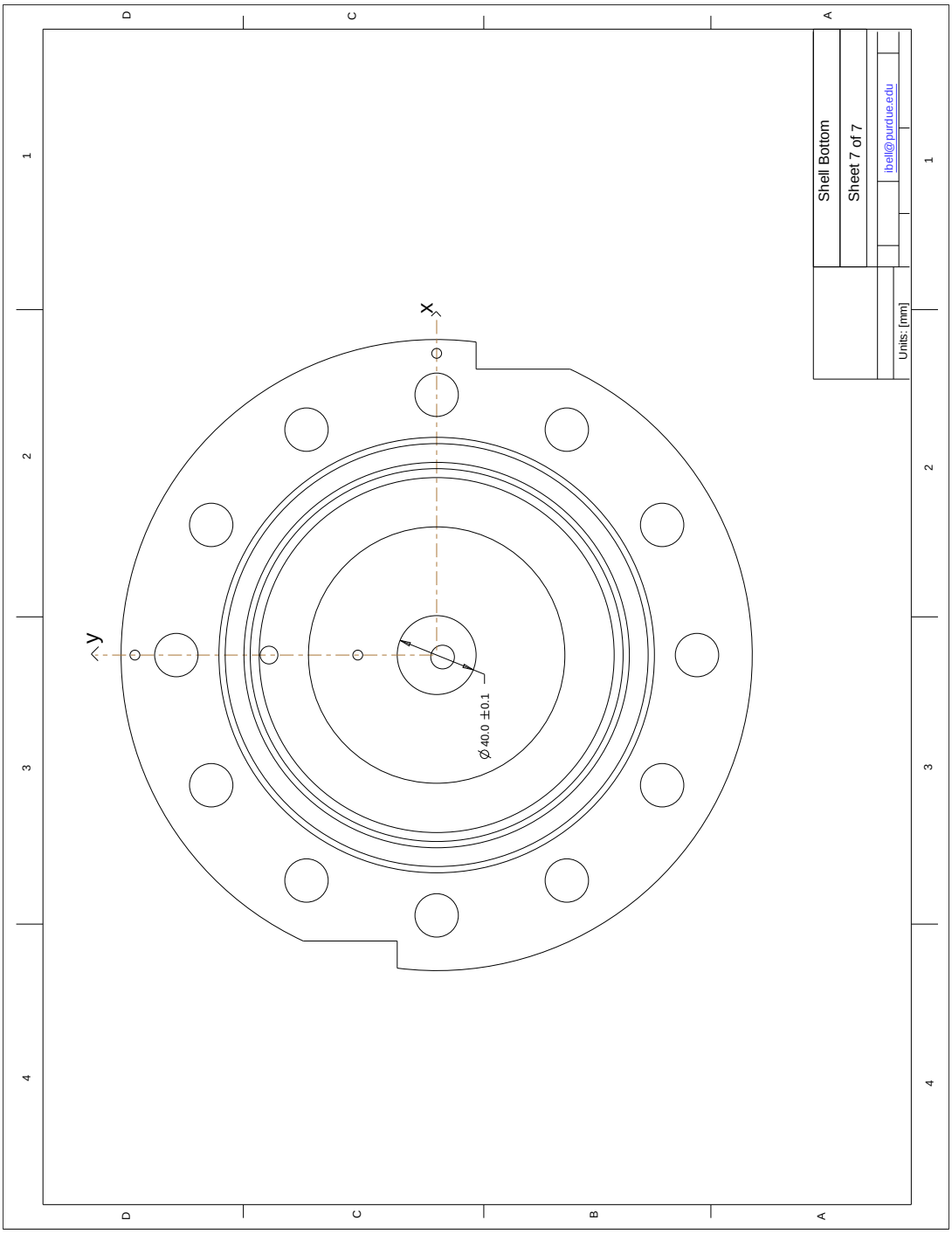


Figure G.1: Continued.

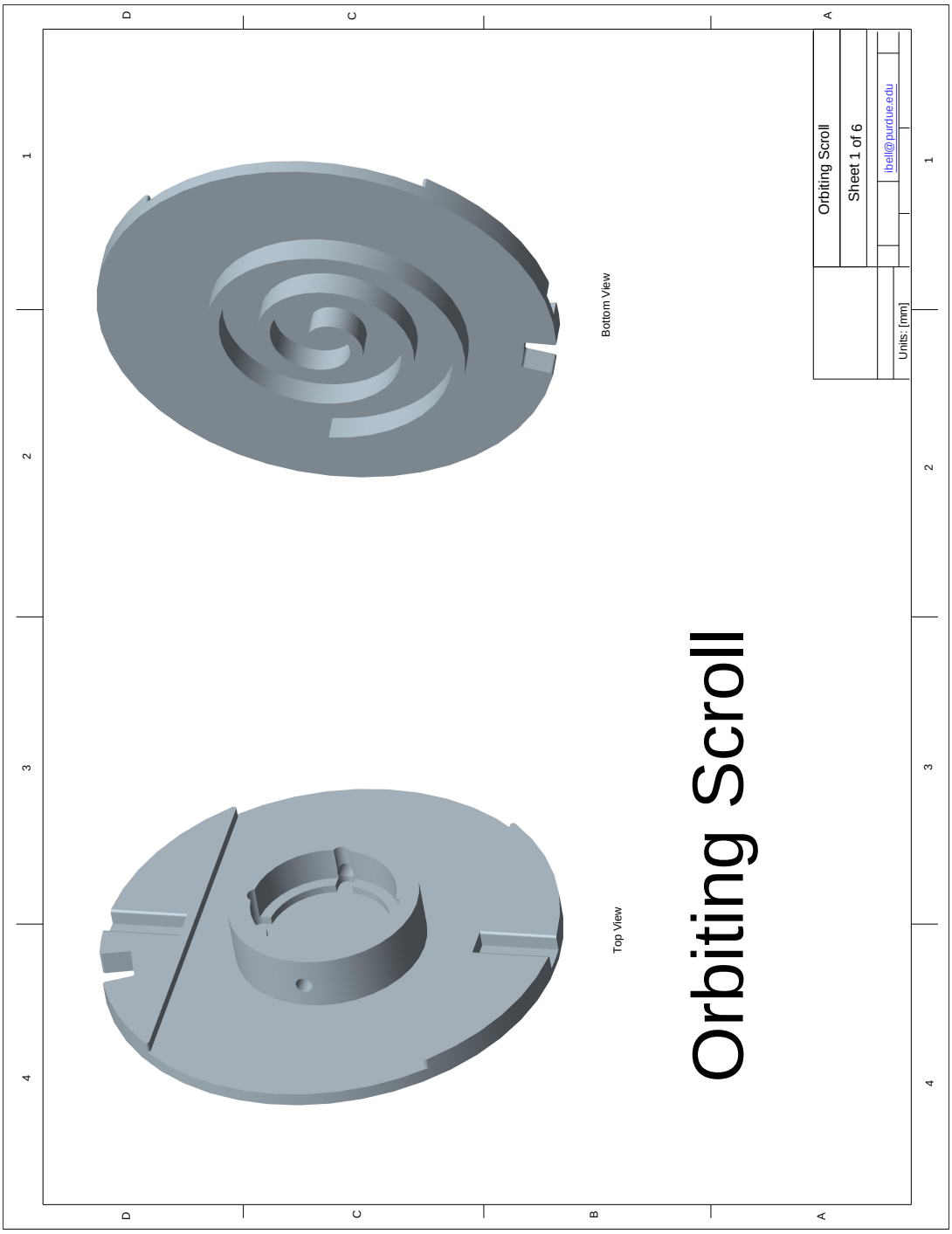


Figure G.1: Continued.

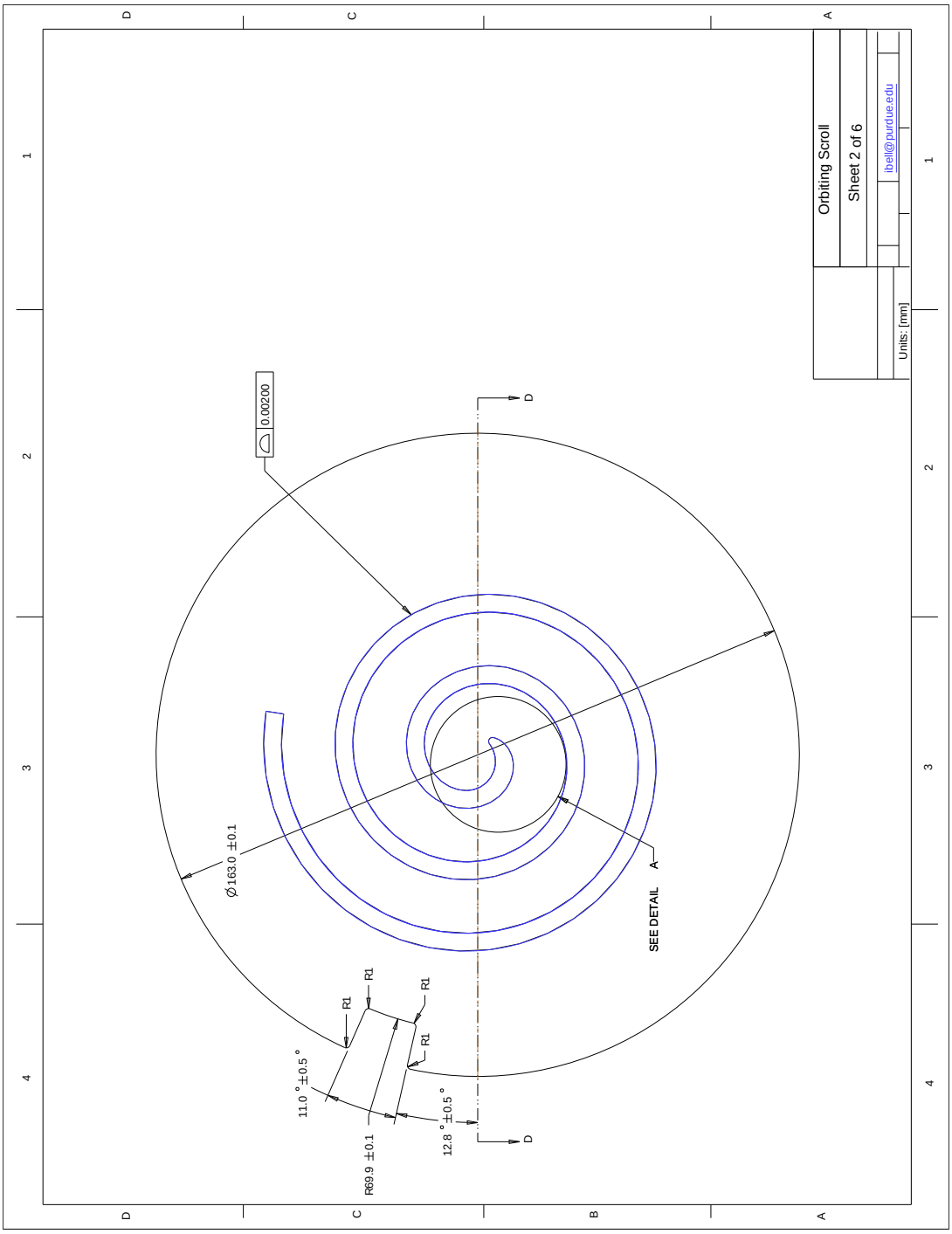


Figure G.1: Continued.

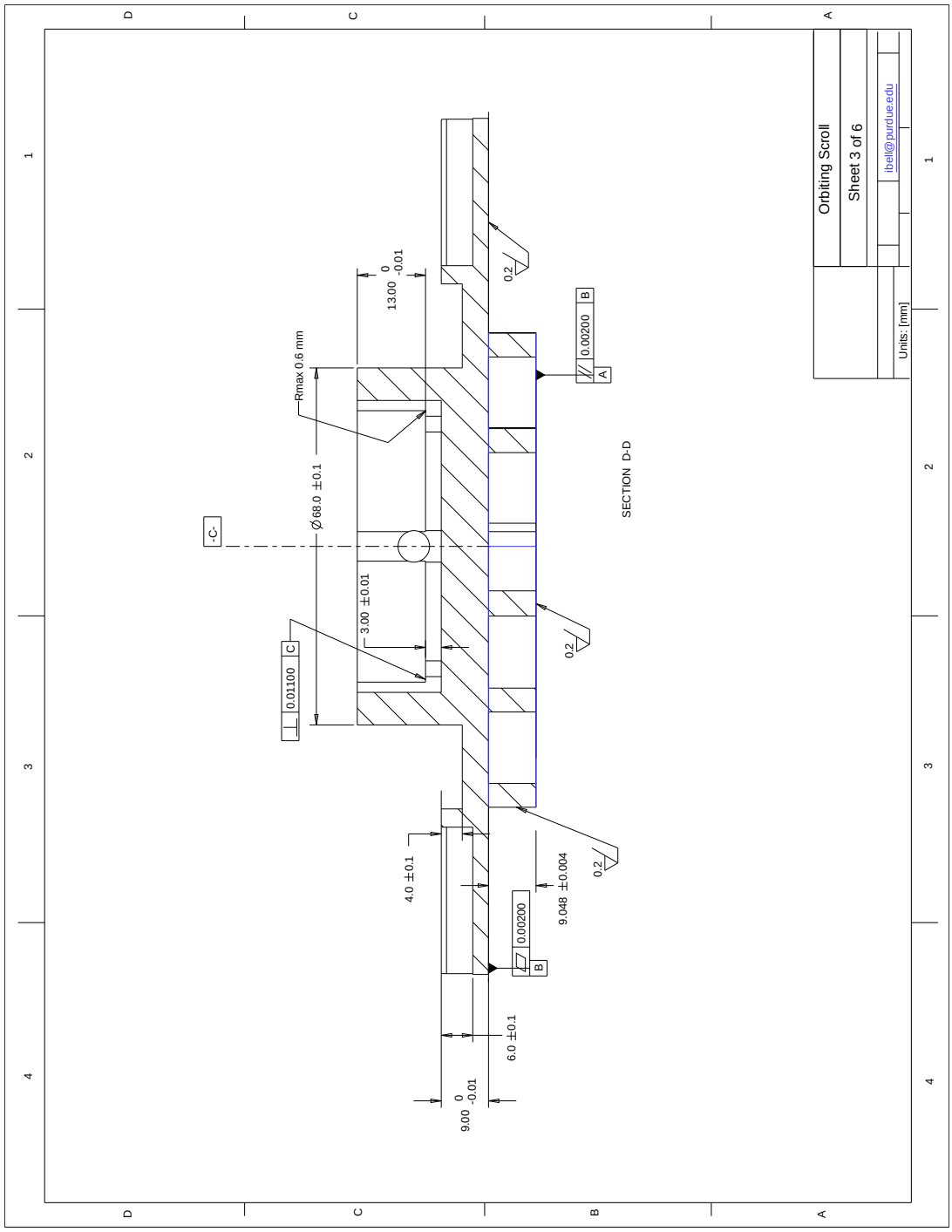


Figure G.1: Continued.

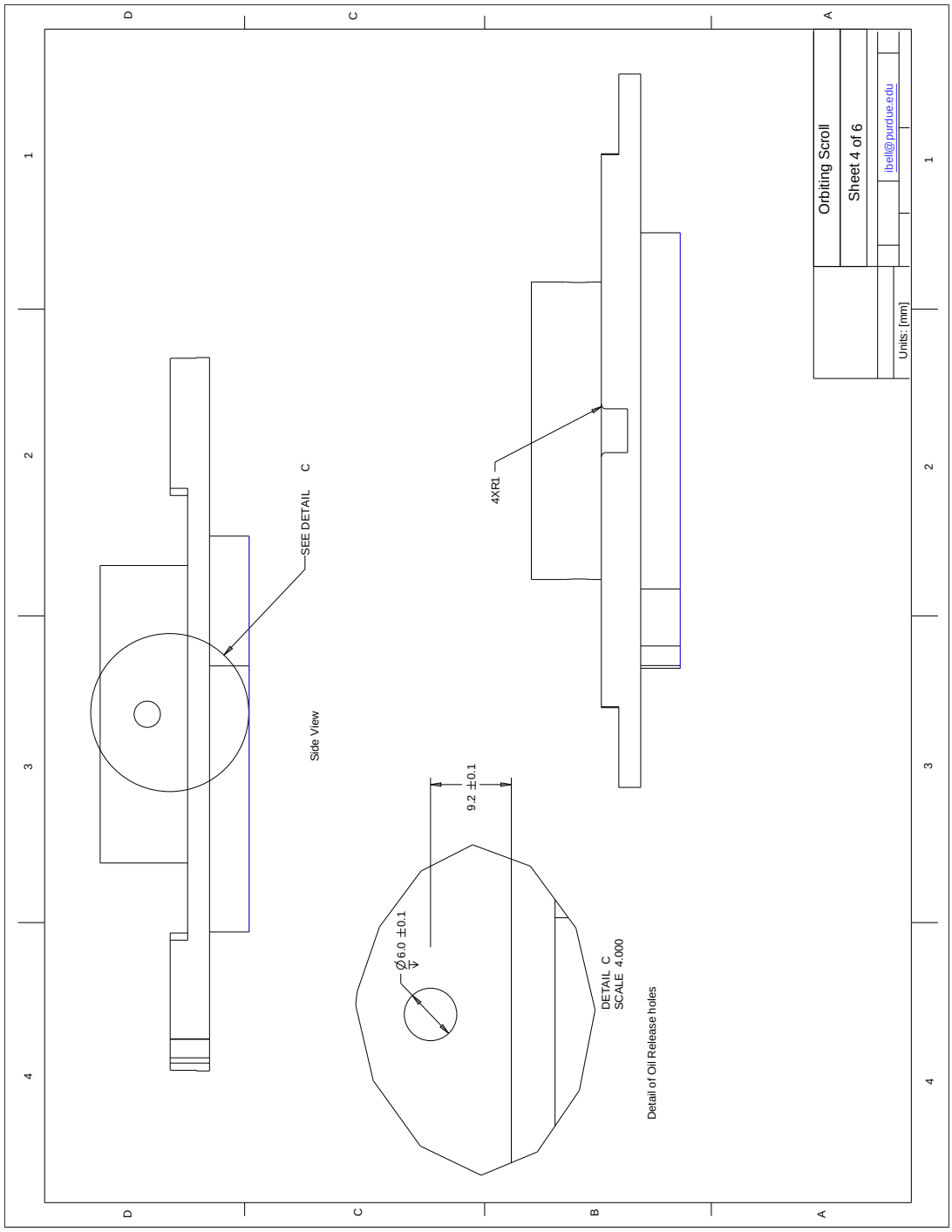


Figure G.1: Continued.

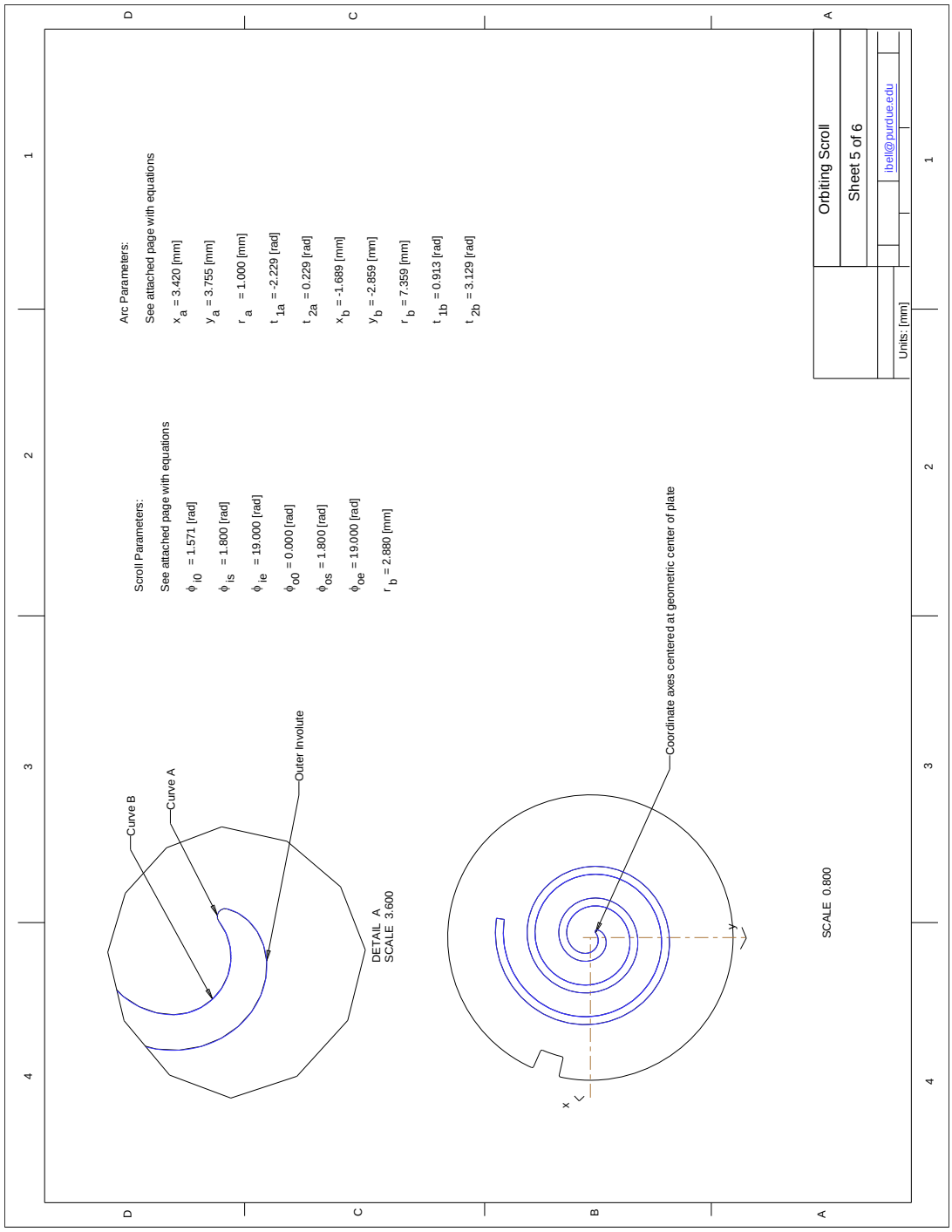


Figure G.1: Continued.

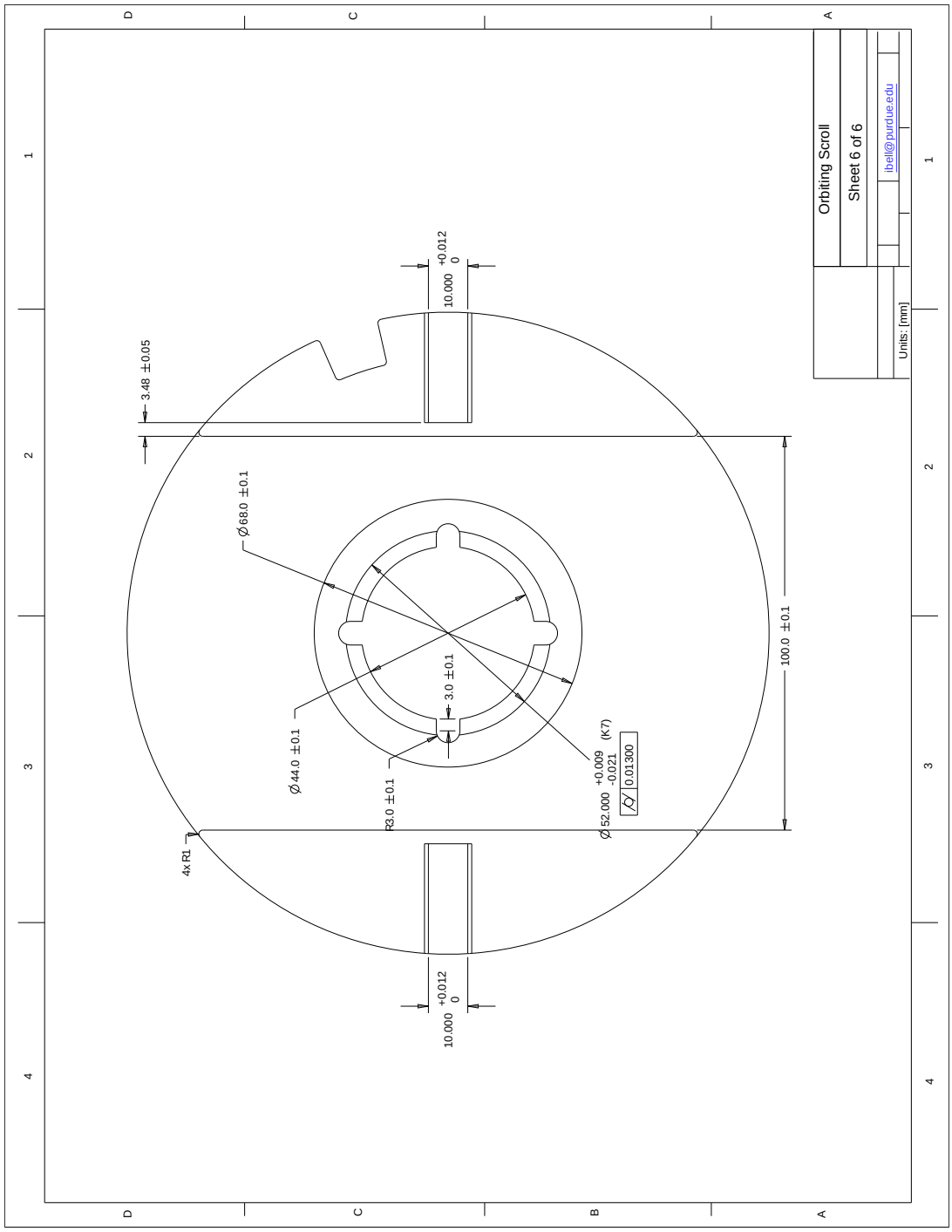


Figure G.1: Continued.

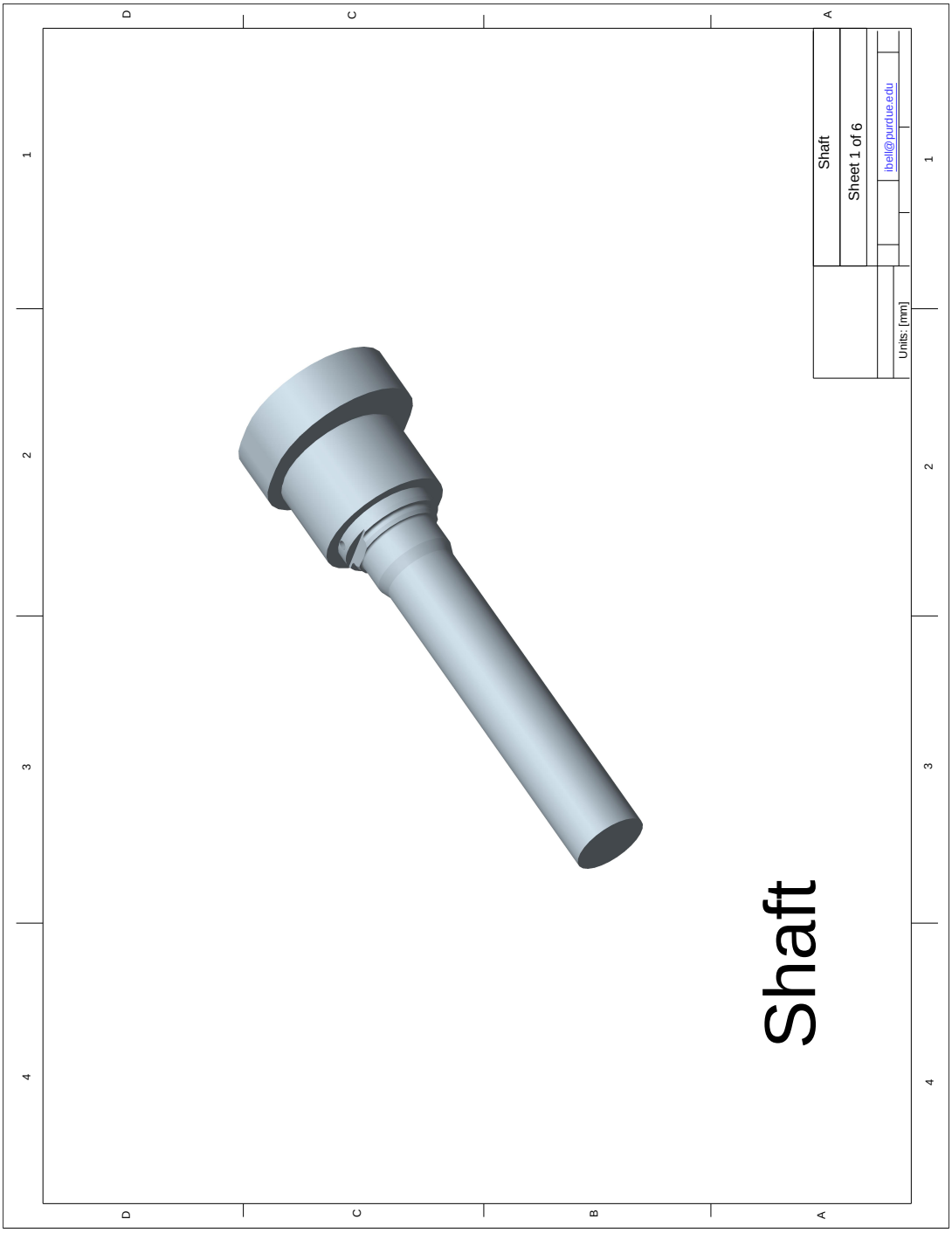


Figure G.1: Continued.

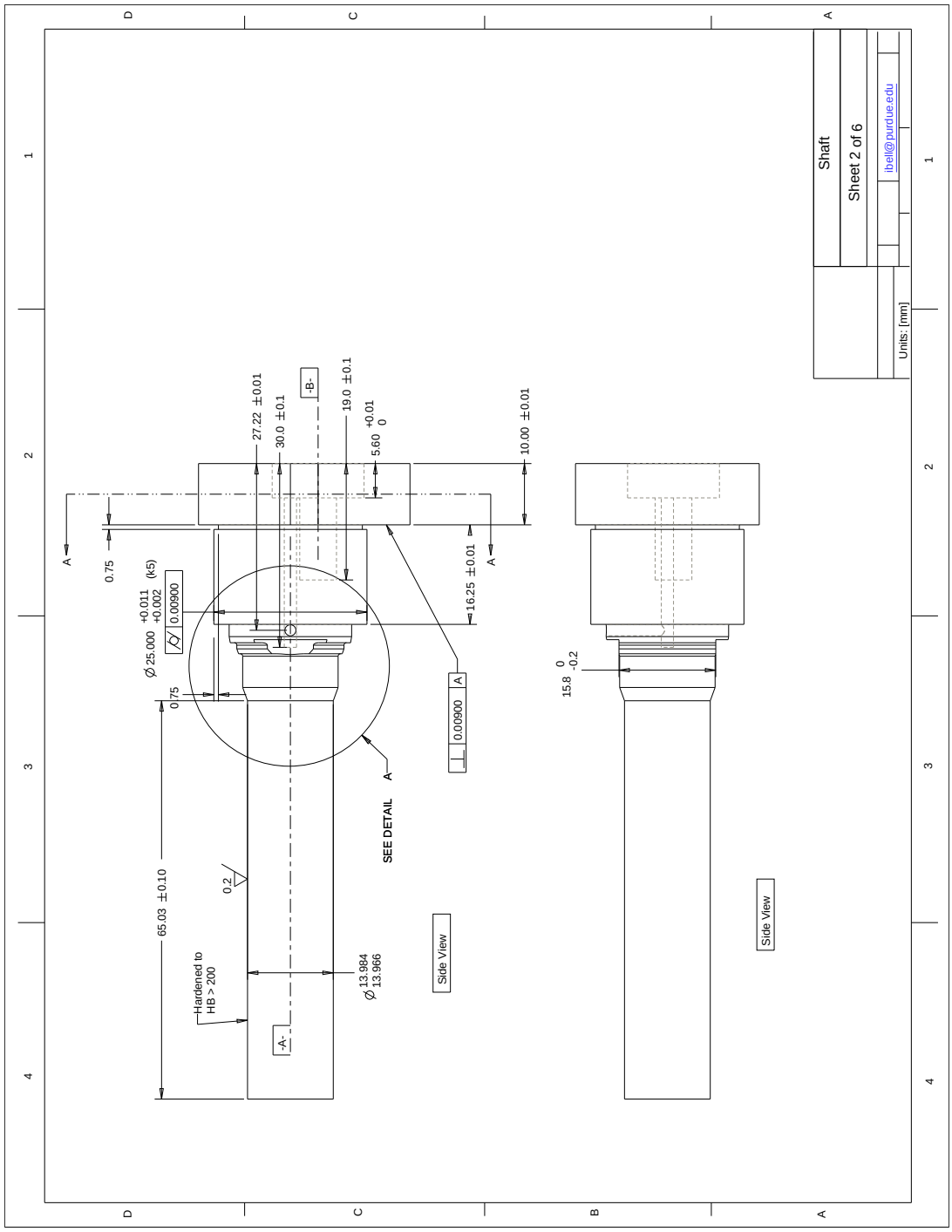


Figure G.1: Continued.

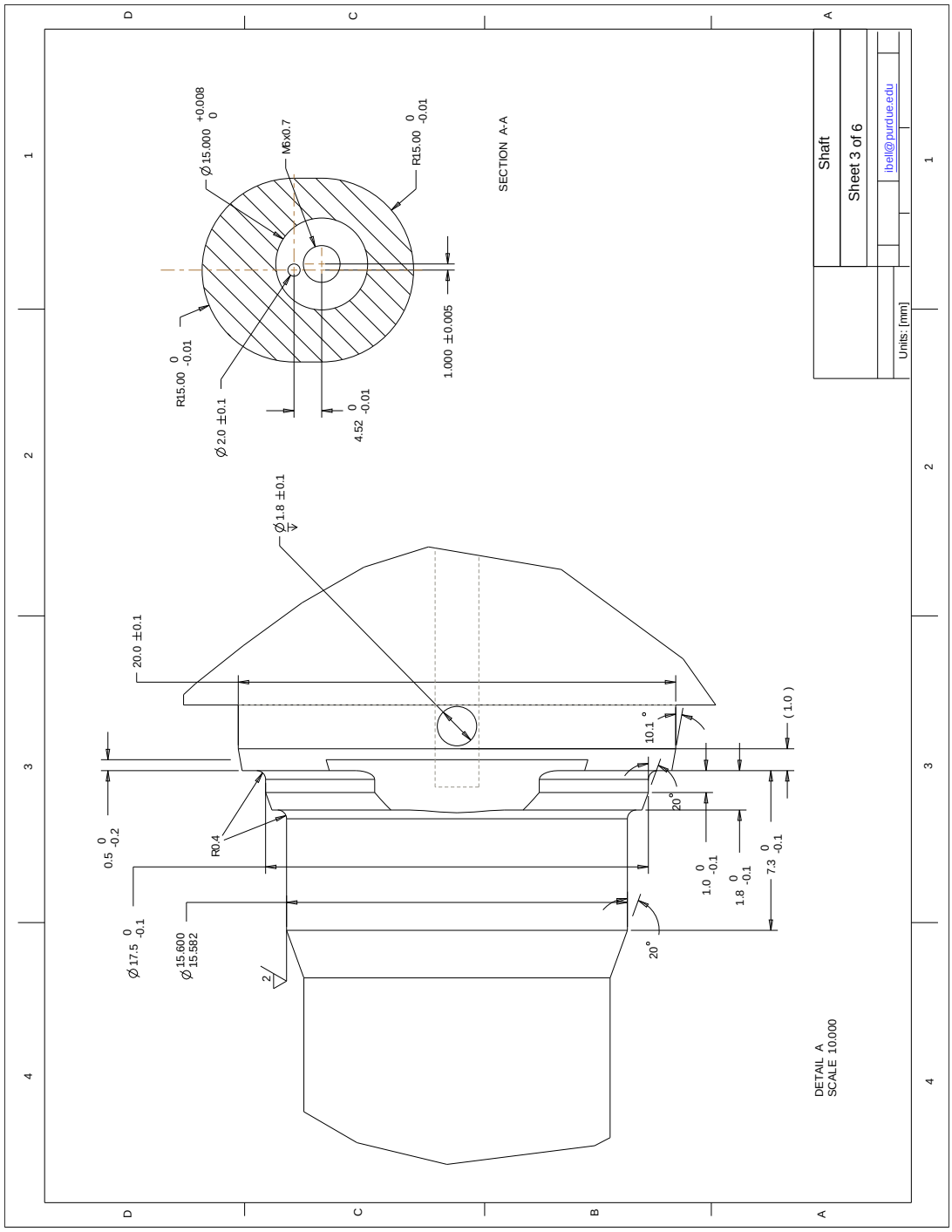


Figure G.1: Continued.

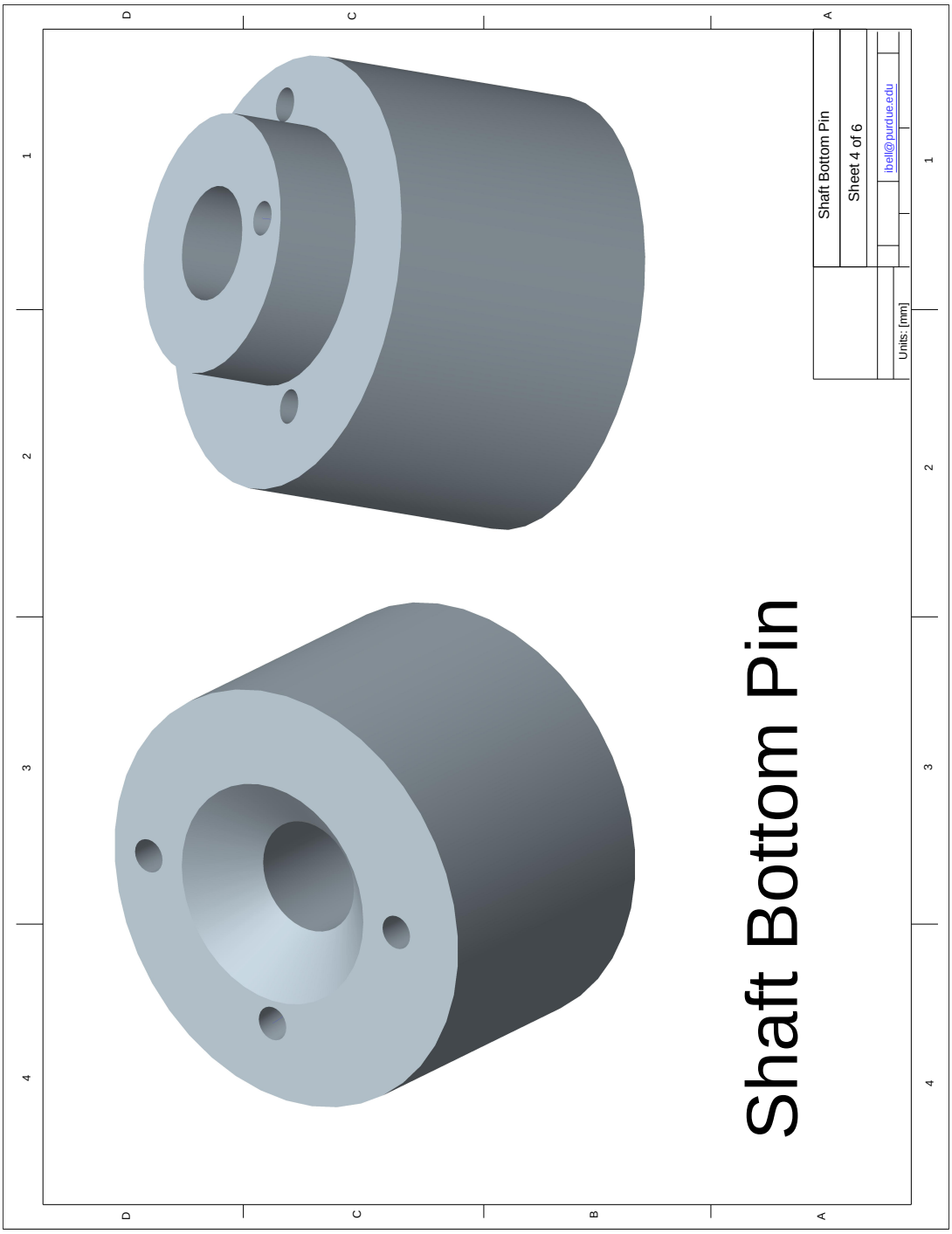


Figure G.1: Continued.

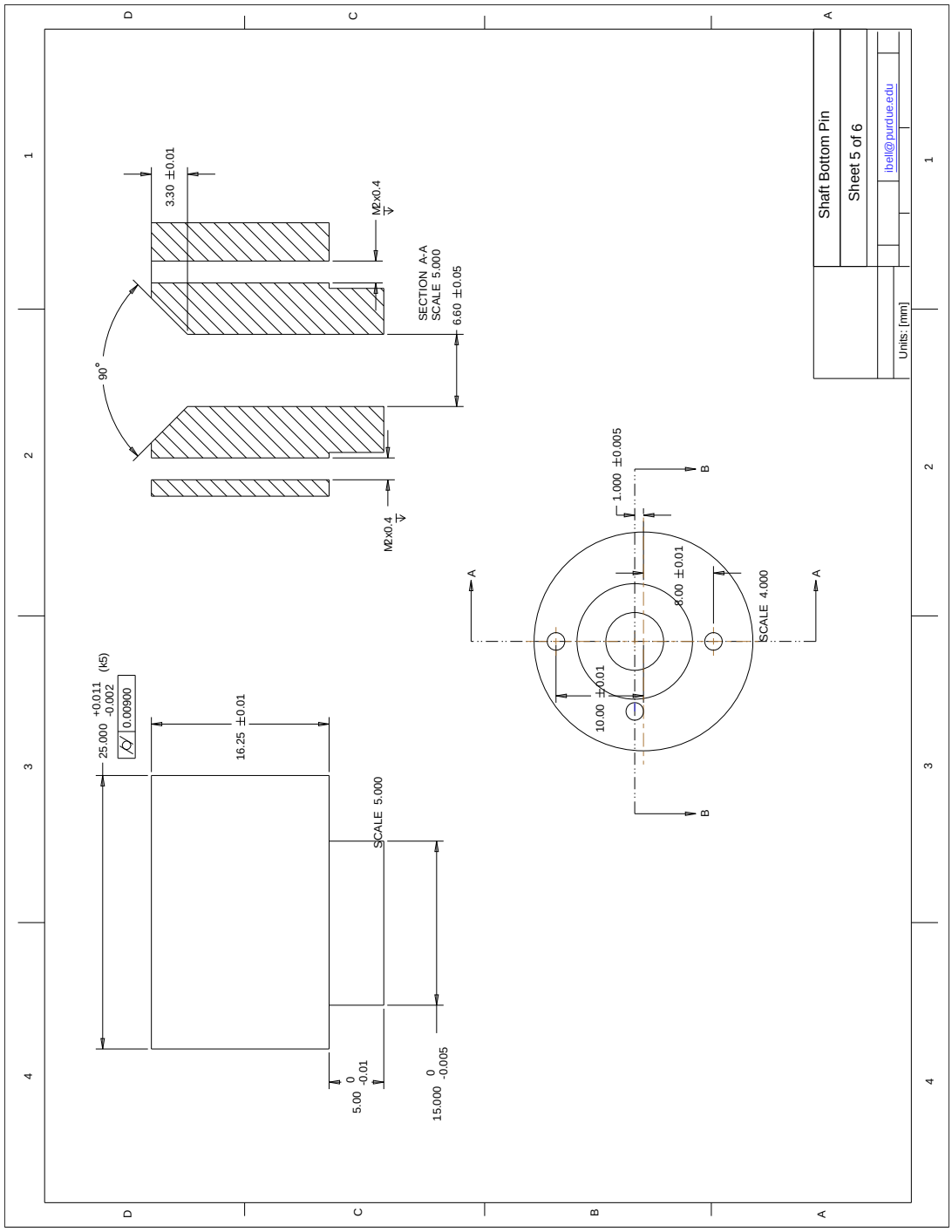


Figure G.1: Continued.

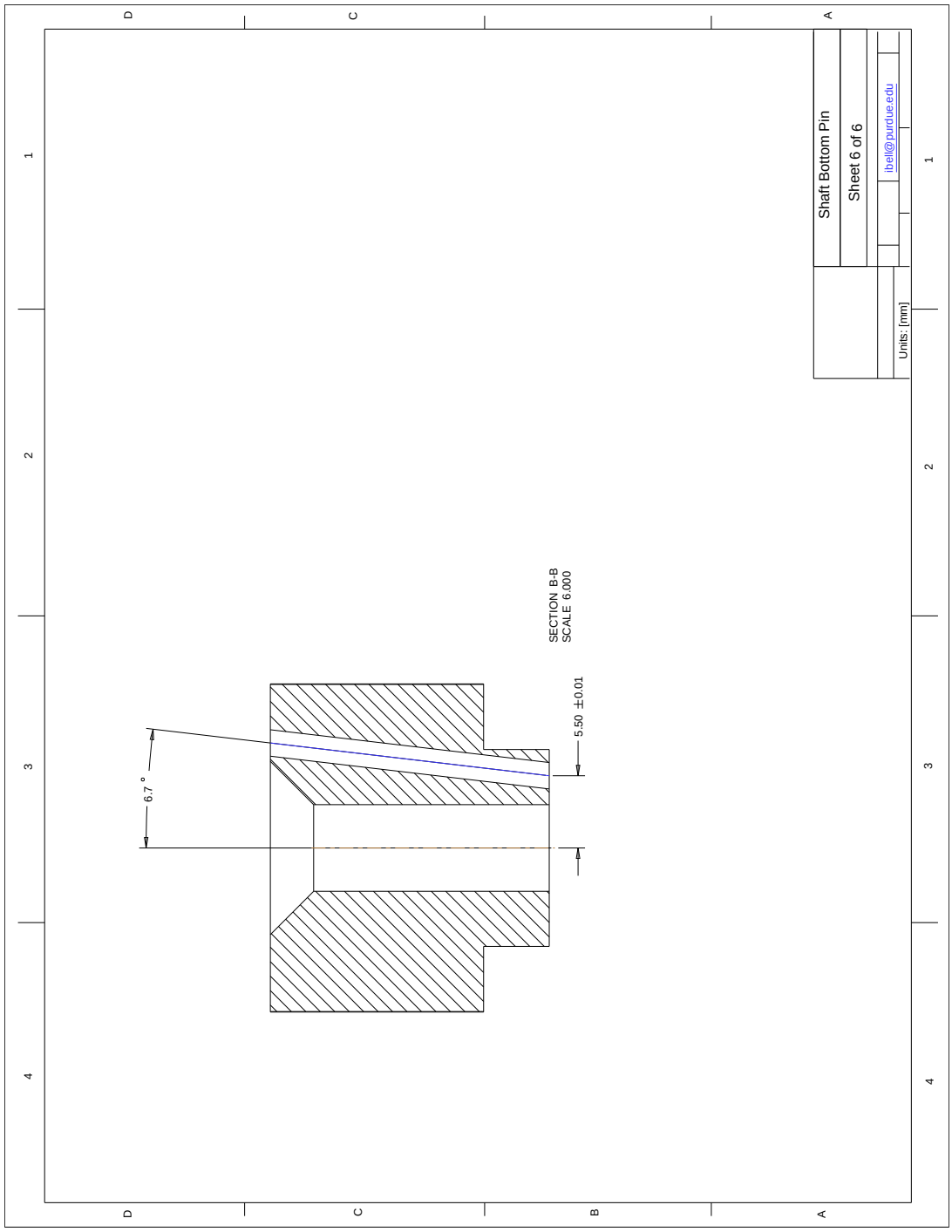


Figure G.1: Continued.

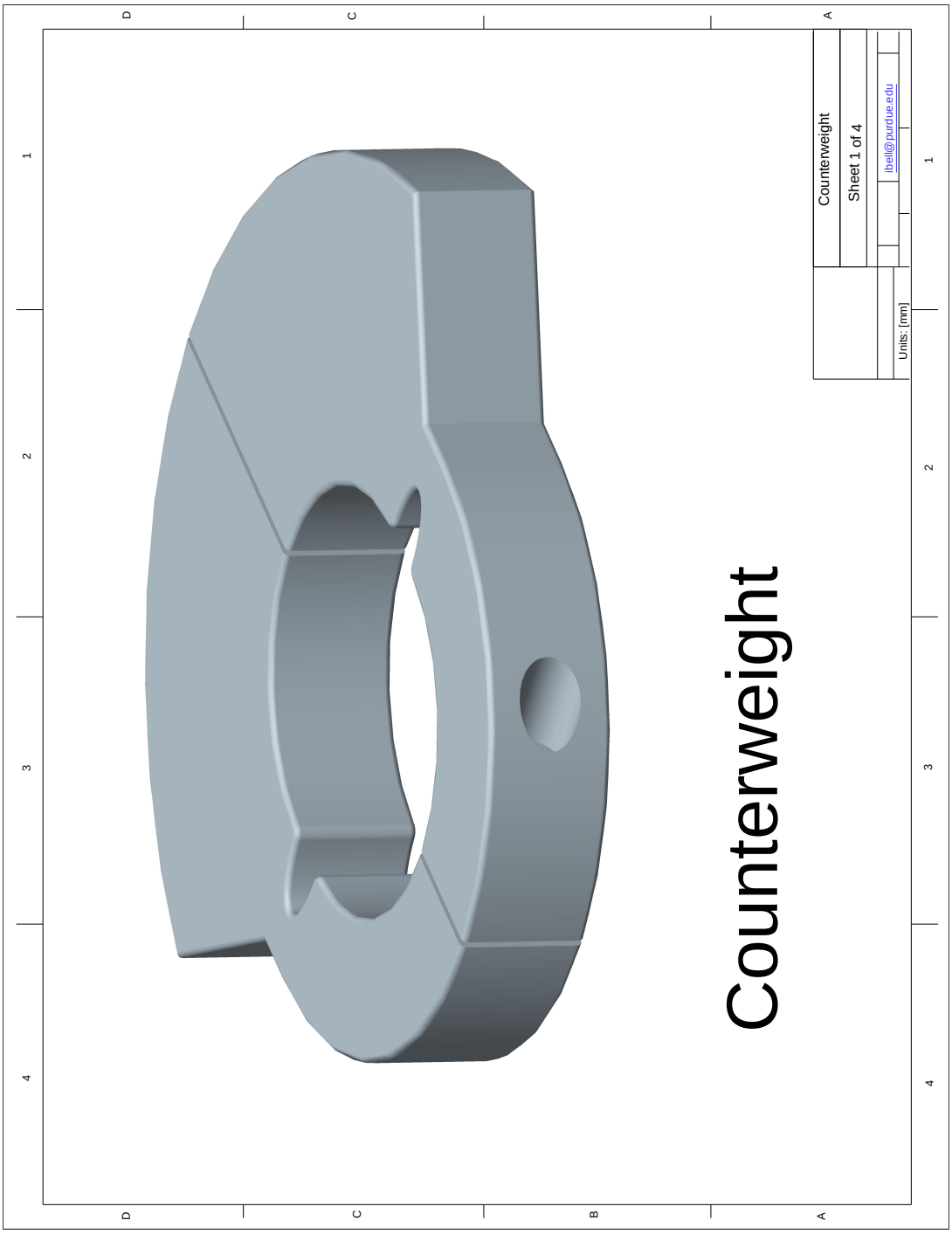


Figure G.1: Continued.

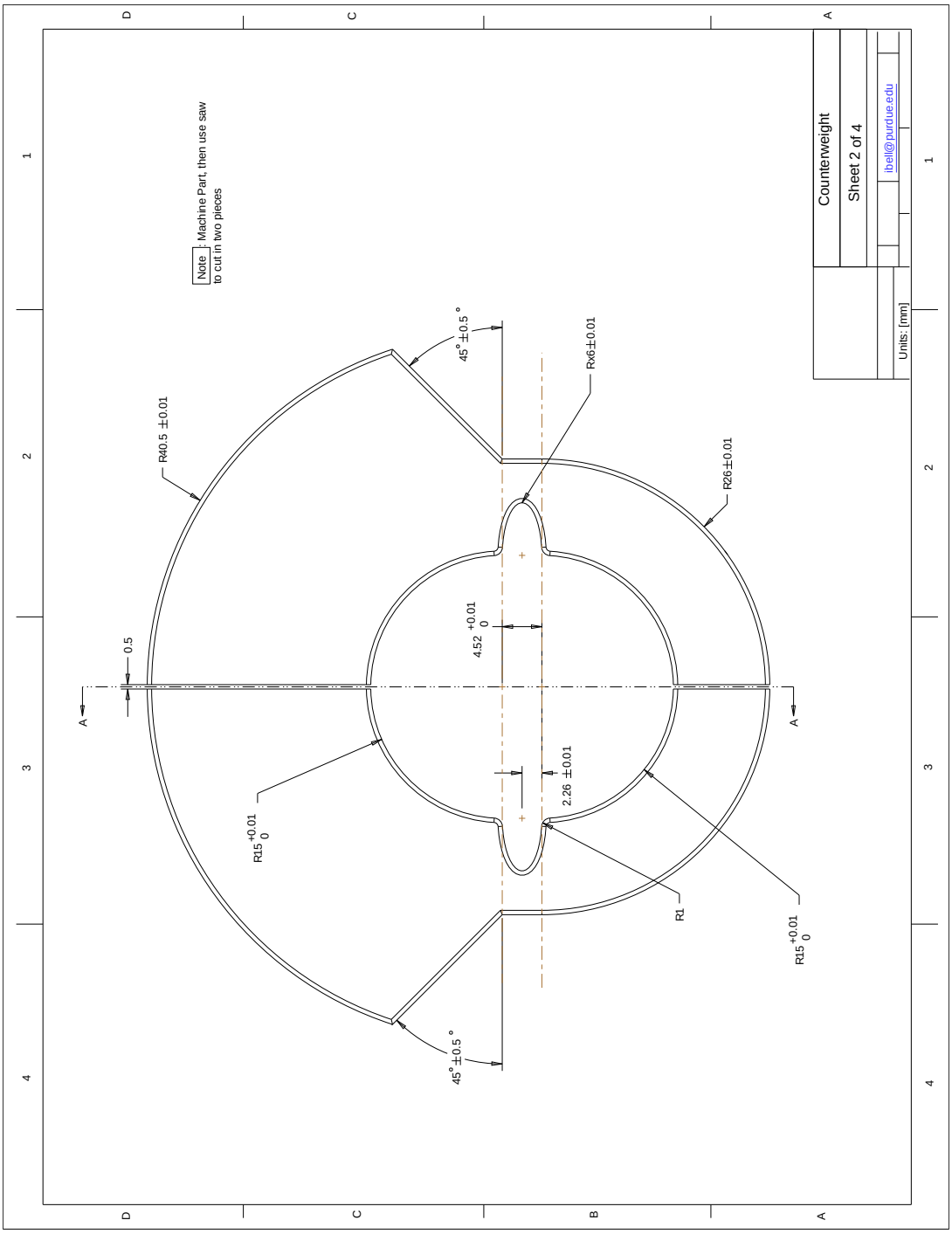


Figure G.1: Continued.

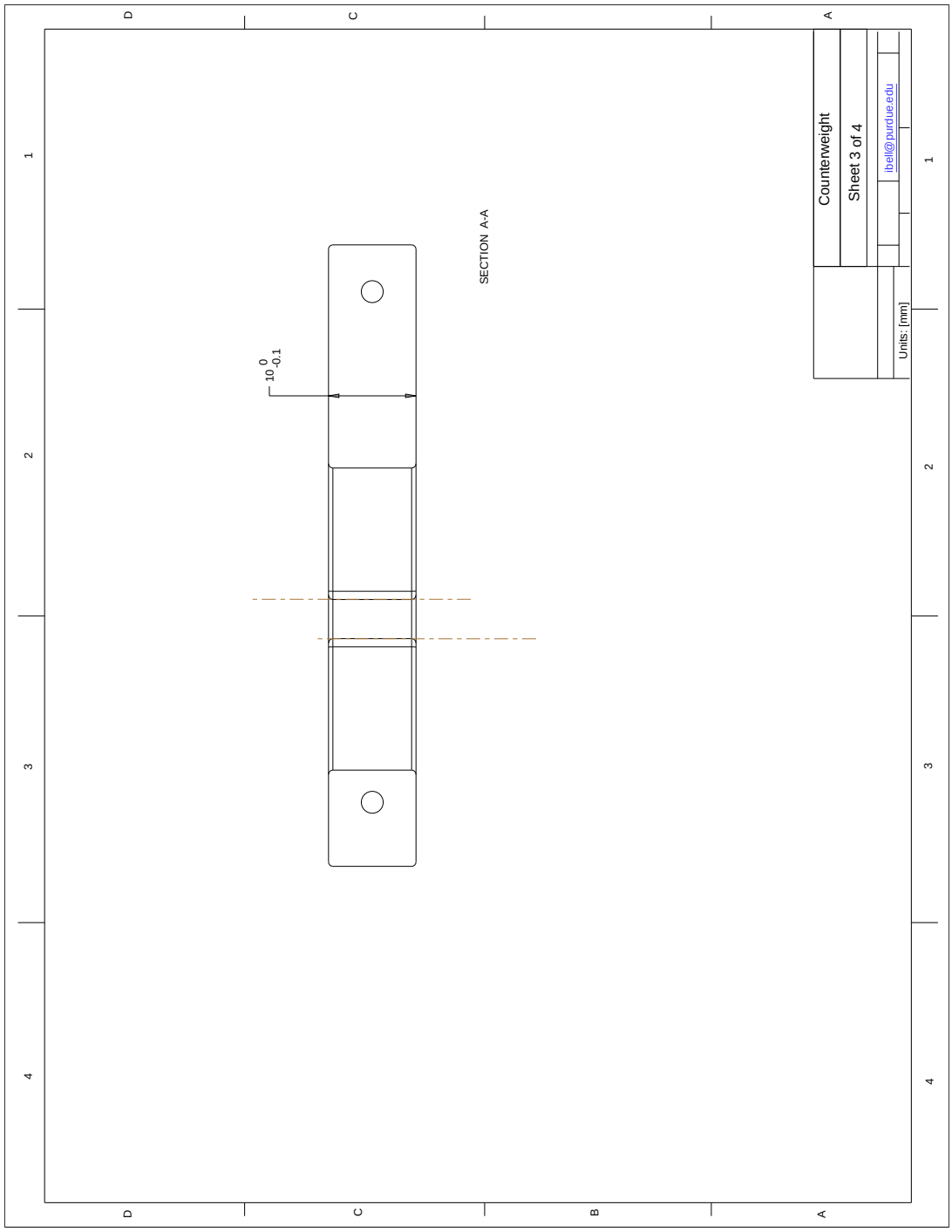


Figure G.1: Continued.

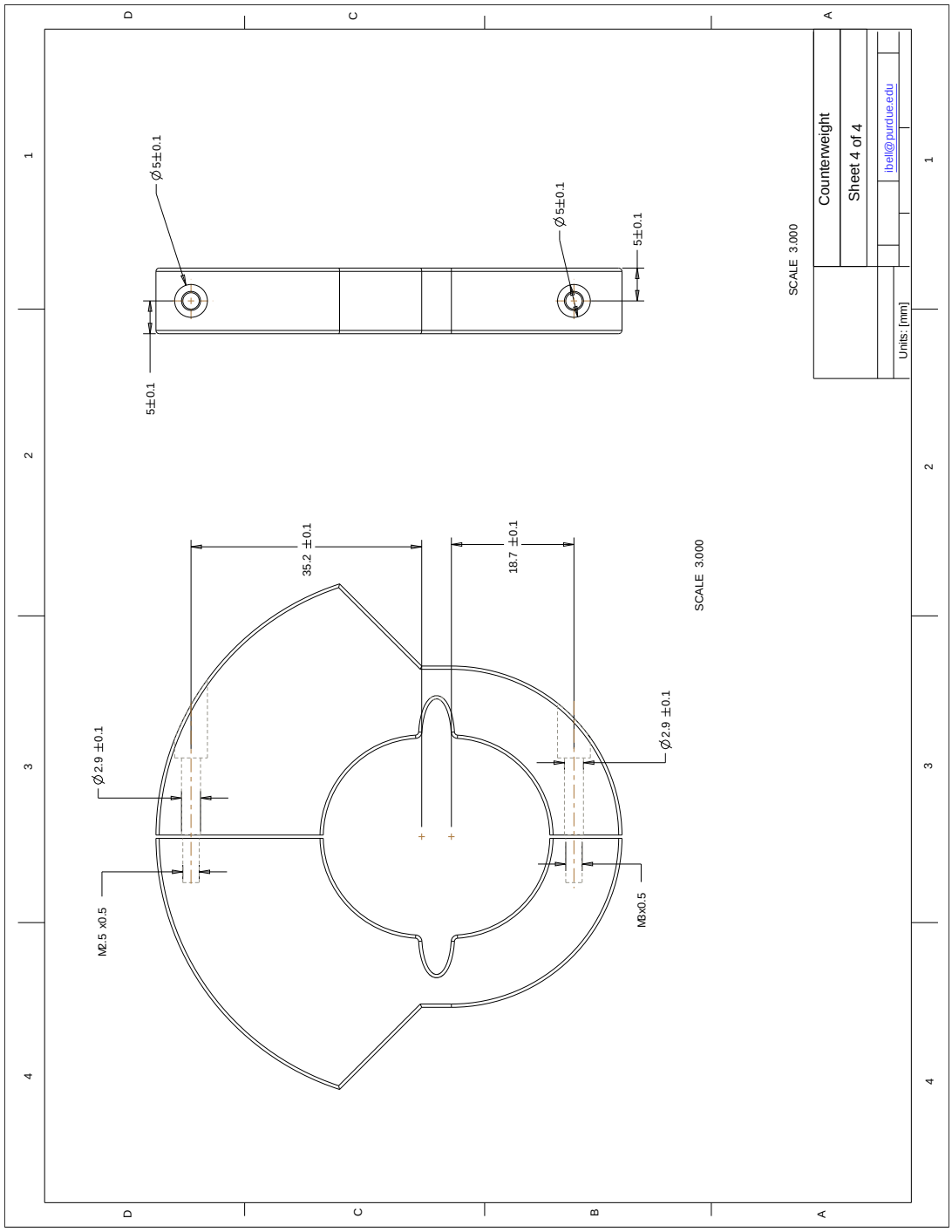


Figure G.1: Continued.

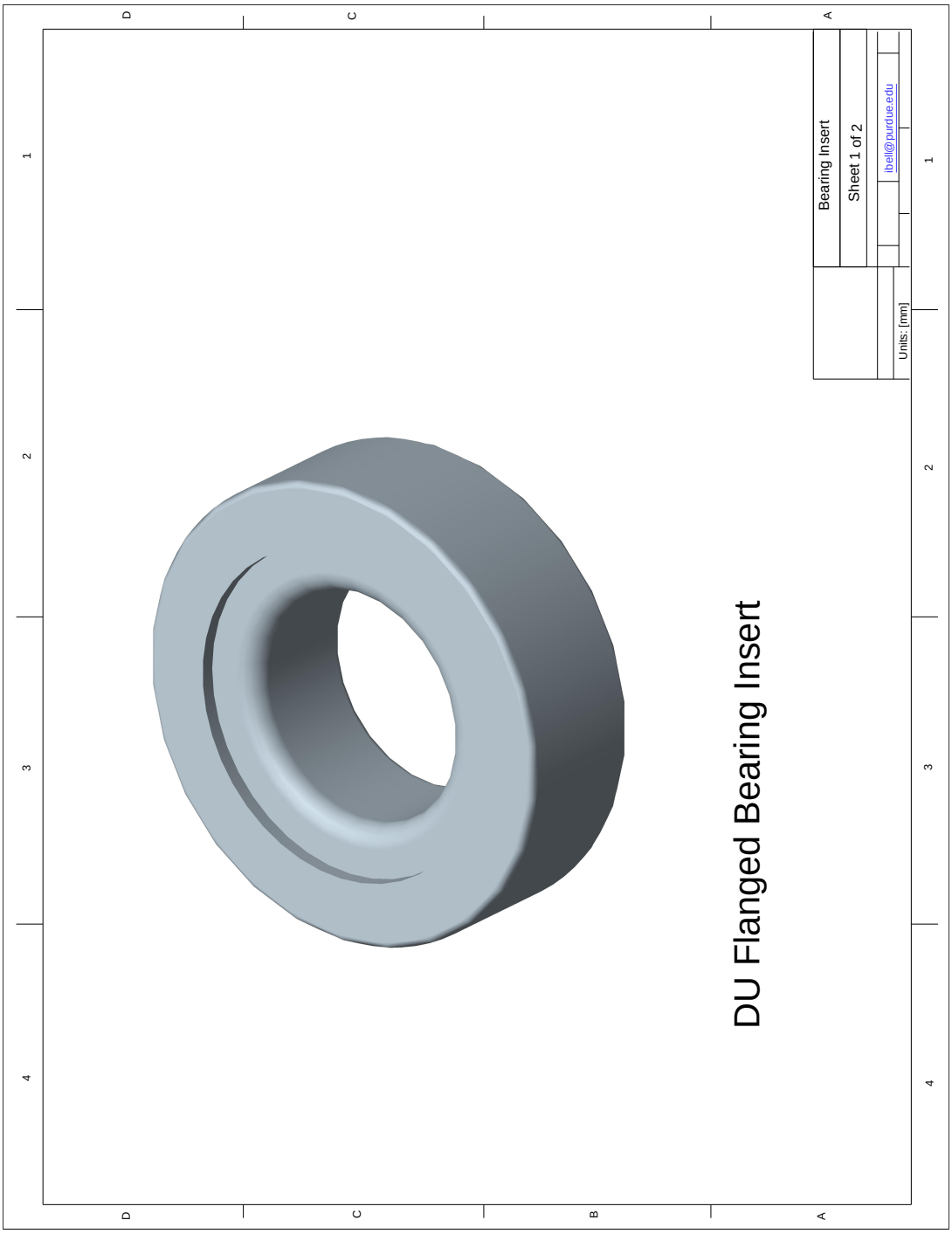


Figure G.1: Continued.

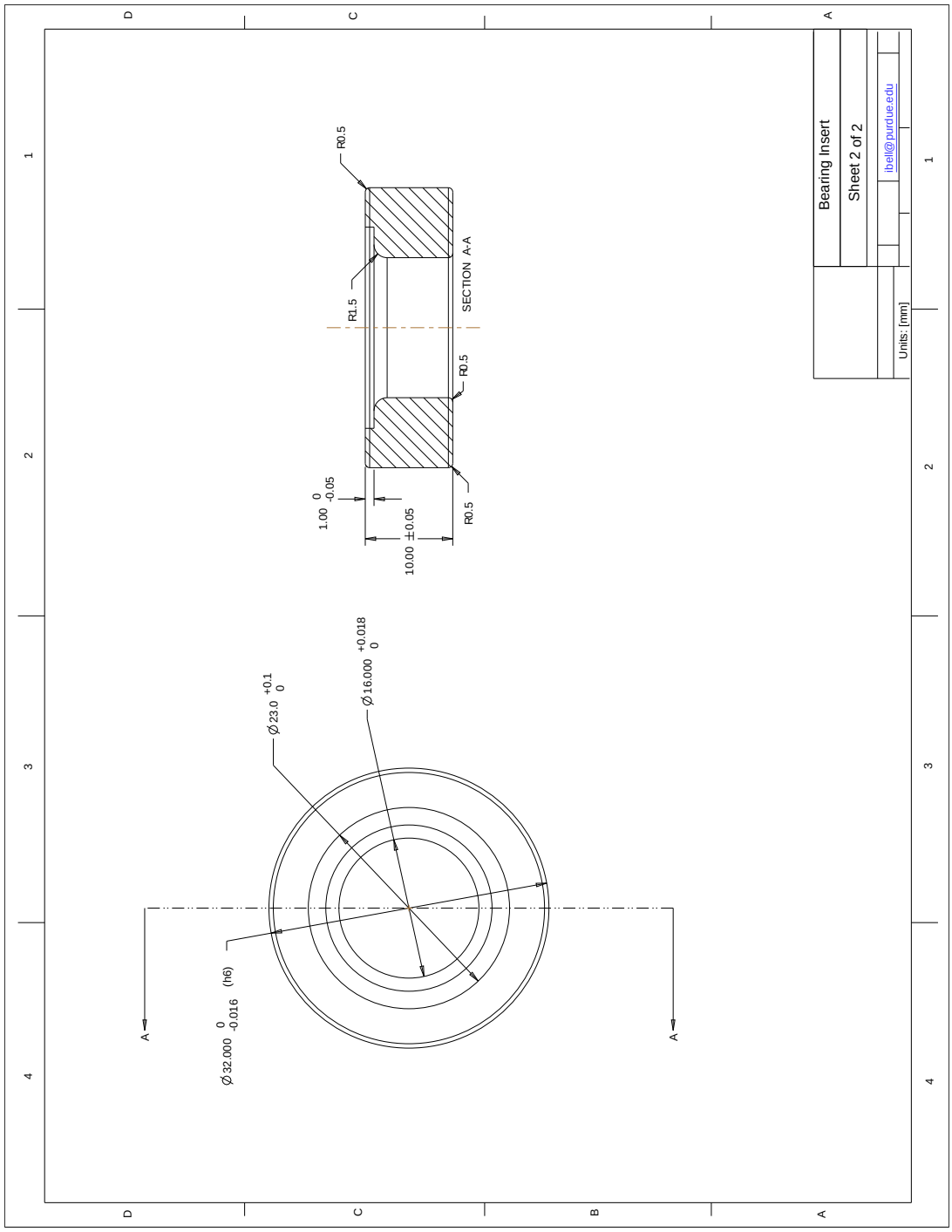


Figure G.1: Continued.

VITA

VITA

Ian Bell was born in Glasgow, Scotland, on October 8, 1983, the son of Duncan and Sarah Bell. He received his Bachelor's of Mechanical Engineering *cum laude* from Cornell University in May 2006. Since then he has been working as a PhD. student in the Ray W. Herrick Laboratory of Purdue University in West Lafayette, Indiana, USA.